Project 3 Reflection: Knowledge-Based AI: Cognitive Systems CS-7637

Saad Khan

April 17, 2016

1 Introduction

This report covers the design and implementation of an AI agent that attempts to solve the 2 x 2 and 3 x 3 Raven's Progressive Matrices (RPM) problems. These problems are image based tests that are used for human IQ determination and are presented in n x n visual matrix form where the object at the last index is to be determined. Typical problems are like the ones shown in Figure 1.

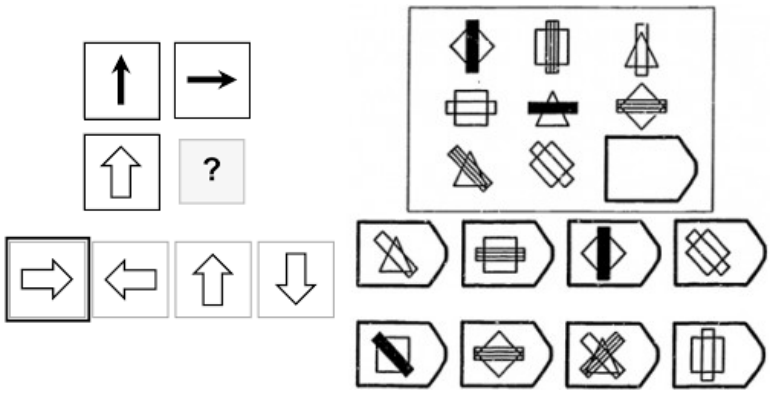

Figure 1: [LEFT]: 2 x 2 Raven's Problem [RIGHT]: 3 x 3 Raven's Problem

2 Problem Reasoning Process

The AI agent uses a combination of semantic network representations and 'generate & test' method to solve the problems. There were 48 problems as part of this project, out of which 12 were 2 x 2 problems from set B, continued from project 1 and the remaining were 3 x 3 problems from sets C, D and E.

2.1 Reasoning Approach

Agent's reasoning is solely based on visual inputs, although, initial design for project 1 was based on verbal approach but was switched to visual in order to apply visual problem solving techniques.

2.2 Criteria for Comparison of Images

The three comparison techniques used to solve the RPMs are shown in the table below:

METHOD	CATEGORY	EXPLANATION
Teversky's Index	2 x 2, 3 x 3	Primarily, similarity was measured using Tversky's index[5]. Tversky's index is the generalized form of Dice's coefficient[6] as show in Figure 2.
RMS Difference	2 x 2, 3 x 3	Root Mean Square (RMS) difference[7] was used for the more difficult problems. Methodology for similarity was a little different for 2 x 2 and 3 x 3 RPMs.
Black Pixel Difference	3 x 3 only	This methodology was only used for 3 x 3 problems by simply computing the count of black pixels in an image while comparing it to other images.

A	1	0	1	1	1	0	0	0
B	1	1	0	1	1	0	1	0

$$\frac{2*bothAB}{onlyA+onlyB+2*bothAB} = \frac{2*3}{1+2+2*3} = \frac{6}{9} = 0.666$$

Figure 2: Example of Tversky's index as generalized form of Dice's Coefficient

2.3 Assumptions/Possible Risks

Following are the basic assumptions and the possible risks that the agent might encounter during problem solving for both 2 x 2 and 3 x 3 cases:

- Instead of creating semantic representations from scratch and then applying generate and test, one of the assumptions was that limited types of representations are already present such as identity transformation, reflection transformation, rotation transformation, XOR and RMS difference. As these are just few techniques, the agent might not generalize well by not exploring other various representations or useful transformations that could exist.
- Another assumption is that horizontal transformations in the top row are analogous to other rows and similar is true vertically and diagonally. Again, based on the limited number of transforms/comparisons being deployed the agent might choose row over column transform depending on the results of the Tversky's index, RMS difference, etc which might not help in the long run when solving difficult RPMs.

3 Problem Solving Process

3.1 Process Flow

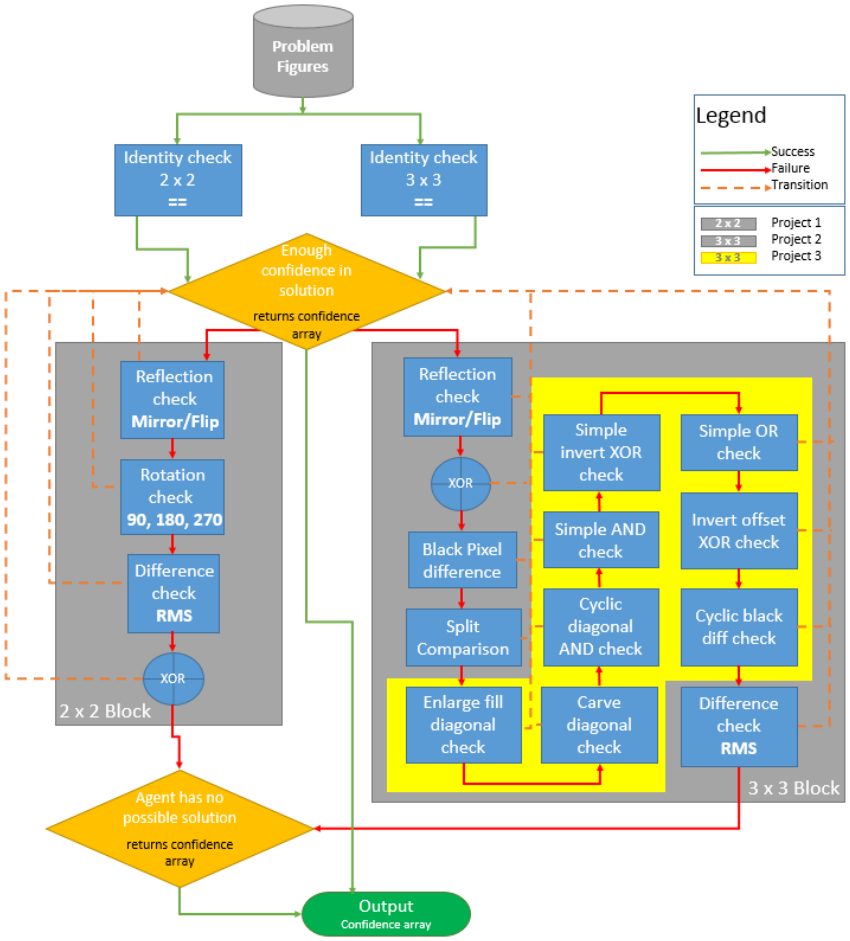

Figure 3: Flow diagram that agent used for problem solving

This section covers the design thought process of the agent followed by the actual implementation process. The agent splits the problems into 2 x 2 and 3 x 3 in the beginning and executes respective process flows as shown in Figure 3. During the process flow, where ever the agent does not have enough confidence towards a particular answer choice, the agent moves onto the next stage otherwise it outputs the answer in form of a list with correct answer having normalized score of 1.0 and all other options having normalized score of 0. As a last resort when all these stages do not give significant improvement and no single correct solution is found, the agent completely gives up and just outputs whatever confidence array it is left with.

NOTE: The yellow highlighted part of the 3 x 3 block shows the techniques implemented specifically for project 3. Another thing to notice is that, RMS difference check is not a part of the yellow highlighted section, as it was part of project 2. I bumped it down at the end only because it was messing up the scoring for most of the other problems in set D and E.

2 x 2 Problems : 5 stages to solve all the problems with most of them using Tversky's index.

3 x 3 Problems : 14 stages in all (as shown by the snake-like process flow) to solve the problems in set C, D and E mostly using a mix of Tversky's index and black pixel difference.

3.2 Scoring, Weights and generation of the Confidence array

After each transformation or comparison method is performed, an associated raw score is assigned to each solution to the problem at every stage. These raw scores are weighted/scaled at each stage according to the ease of transformation in descending order. Once all 6 options for 2 x 2 and all 8 options for 3 x 3 problems have been assigned weighted raw scores the result is passed through the rank method. This method assigns all of its confidence to scores that are greater than or equal to 95% of the maximum raw score in the array. Figure 4 shows 2 such examples, which the agent considers as the hardest problems as it has to pass through all the stages of the process flow to check confidence.

```
--------------------------------
    Challenge Problem B-05 2x2
--------------------------------
[13.77807540881784, 13.908724793596754, 21.548048240298655, 23.39697876826809, 13.367251173129167, 23.622697874339213]
[0, 0, 0, 0, 0, 1.0]
normalized confidence: [0.0, 0.0, 0.0, 0.0, 0.0, 1.0]

--------------------------------
    Challenge Problem B-02 2x2
--------------------------------
[15.388781104500815, 13.93206124446213, 15.382982040924622, 13.759346417056316, 15.362753106854882, 13.782734900700838]
[1.0, 0, 0.6231629044033608, 0, 0, 0]
normalized confidence: [0.6160811076246091, 0.0, 0.38391889237539084, 0.0, 0.0, 0.0]
```

Figure 4: raw score ranking examples

If we look at the results for challenge problem 05, after applying all the different techniques the agent had assigned all 6 answer options certain raw scores. The array at the end is then passed to the rank method with threshold of 0.999. The rank method then only considers a continuous confidence window of [max(raw score) - 0.999 * max(raw score)]. If there are other scores in that window, the rank method will assign appropriate percentage weight and if not then it assigns all its confidence to the maximum raw score. Then that array is passed to the normalizing method which ultimately gives all of its confidence to answer option 6, which in this case is the correct answer.

On the other hand, for the example taken from challenge problem 02, the agent again passes through all the stages ending up with the raw score array looking like the one shown in Figure 4. The raw scores assigned to answer options 1 and 3 are so close that even when a threshold of 0.999 is applied to the rank method it is unable to put all of its confidence in one answer option. The weighted score that is passed to the normalizing method is [1.0, 0, 0.6231629044033608, 0, 0, 0] and when it is normalized, the agent has shared confidence in 2 answer options 1 and 3, where in fact 1 is the correct answer. In this case agent only gets the answer partially correct.

3.3 Agent Implementation

This section covers the actual problem solving method applied at each stage by the agent.

3.3.1 Stage Description for 2 x 2 Problems

STAGE	EXPLANATION
Identity, Reflection & Rotation	For identity: if 2 images being compared are exactly identical, an if/else block increments the raw score accordingly. For reflection: Agent flips the image to be compared, either horizontally or vertically, checks equality of images and then assigns a raw score accordingly. Rotation check is a little different where the agent considers rotation by 90, 180 or 270 degrees. Horizontal similarity of image A with rotated image B is compared to the similarity of image C with rotated version of each of the answer options. With an if/else block, vertical similarity i also checked. The higher the similarity, higher the raw score is assigned to a particular answer option.
	Bias/possible mistake : In these checks, agent is giving preference to horizontal comparison over vertical comparison and is also assuming that rotations can only be of 90, 180 and 270 degrees, which could harm the performance of the agent for difficult RPM problems.
XOR check process	This takes in provided images A, B and C along with answer option all at once and computes the Tversky's index after XOR-ing the pair of images. Output is a ratio of the similarity measure XOR_AB to similarity XOR_C'?'and compares it to a threshold (close to 1) to assign a raw score to a particular answer option.
	Bias/possible mistake : Agent is biased towards horizontal comparison and also the threshold value is just a calculated guess that satisfies basic set B and might not work for difficult problems.
RMS Difference Check	Here the agent compares the absolute difference, horizontally for AB and C'?' and then vertically for AC and B'?', within a certain threshold. If the difference is less than 5 the agent assigns some score to it.
	Bias/possible mistake : Again the agent is biased towards horizontal comparison and also the threshold value is just a calculated guess that satisfies basic set B and might not work for difficult problems.

3.3.2 Stage Description for 3 x 3 Problems

STAGE	EXPLANATION
Identity & Reflection	The agent starts with identity comparison based on Tversky's index. All the similarity calculations are based on the cases shown in Figure 5 i.e. similarity between BC - H?, AC - G?, etc. If the images being compared are same or very similar to each other a raw score is assigned as a multiple of the Tversky's index result. If the agent is unable to solve the problem using identity, it moves onto reflection. In contrast to the 2 x 2 reflection, in the 3 x 3 case, images A and C, D and F are compared with G and '?' for horizontal reflections while images A and G, B and H are compared with C and ? for vertical reflections. Tversky's index is computed and if difference is less than a certain threshold, a raw score as a multiple of the Tversky's index for similarity between G and reflected ? or C and reflected ? is assigned to that particular answer option. Alongwith, horizontal and vertical identity check, diagonal check is also implemented.
	Bias/possible mistake : Again, in this case the agent reasons over horizontal identity and reflection check before it performs vertical comparison, which might work for basic problems but not for advanced problems.
XOR check process	XOR implementation is divided into 2 parts, (i) XOR based on black pixel ratio difference and (ii) XOR based on Tversky' index comparison. The later only solves problem C-12 while the former addresses easier problems. Ratio of XOR_AB to XOR_BC is compared with ratio XOR_DE to XOR_EF and if it is below a certain threshold, ratio of XOR_AB to XOR_BC is further compared to the ratio of XOR_GH to XOR_H ?. If both these differences are below a certain threshold, a weighted raw score as multiple of the following expression is assigned to that answer choice. **(1 - abs(XOR_AB/XOR_BC - XOR_GH/XOR_H ?))** The best answer would have the least difference, so assigning raw score this way, maximizes the confidence in that answer choice. On the other hand, the second implementation of XOR is based on calculating the absolute difference after applying Tversky's index. XOR_BC is compared with XOR_H? and XOR_EF is also to XOR_H?, if the difference in similarity is less than a certain threshold, raw score as multiple of the following expression is assigned to the options. **(1 - abs(Tversky(XOR_BC, XOR_H?) - Tversky(XOR_EF, XOR_H?))**
	Bias/possible mistake : XOR check is only implemented for vertical and horizontal XOR comparison and involves a lot computations before actual comparison checks, which might delay the overall computation time.
Black Pixel Difference Check	Here the agent computes difference of black pixel count ratio for the images AC/AB and DF/DE, if this is below a certain threshold, the agent assigns raw score as multiple of: **(1 - abs(BlackDifferenceAC/BlackDifferenceAB - BlackDifferenceG?/BlackDifferenceGH))**
	Bias/possible mistake : The biggest risk involved here is, I have set a lot of thresholds, which might be solving problems in this Basic Set C but would definitely hinder the performance of he agent for more difficult problems.

Split Comparison Check	Split comparison specifically adresses basic problem C-09. This technique actually splits the images before comparison. There can be many ways, but the basic method implemented was to split 2 images to be compared and find Tversky's measure by swapping the images left with right for comparison as shown in Figure 5 [RIGHT]. The agent checks for the difference of Tversky's index calculated between image Left A and Right C with Right A and Left C. If this value is below a certain threshold, the agent then applies Tversky's index to image Left G and Right '?' or Right G and Left '?' and then the agent simply assigns a raw score as a multiple of the Tversky's index.
	Bias/possible mistake : This split comparison method is only applied for horizontal relationships to address problem C 09 only, however, the agent might not work properly if there is a vertical relationship of this type.
Enlarge Fill Diagonal Check	This technique addresses problem D-08 by diagonally enlarging images G, B and F and generating XOR with images E, A and '?' respectively. Color inverted images of these results (invert_XOR_GE, invert_XOR_FA and invert_XOR_B'?' are then compared against images C, H and D respectively. If the result is close to 1, a raw score is assigned to the confidence array accordingly for each answer choice.
	Bias/possible mistake : Comparion was only performed on Right Diagonal and will not work for left diagonal.
Carve Diagonal Check	This technique addresses problem D-07 and D-10. For D-07, the agent carves out the inner images and compares them Right diagonally while compares outer images Left diagonally. (Right and Left diagonals are shown in Figure 5 [LEFT]). On the other hand, for D-10, the agent again carves out the inner images and compares them Left diagonally while compareing outer images Right diagonally.
	Bias/possible mistake : Comparion was only performed diagonally and would not work for problems with vertical or horizontal relational pattern.
Cyclic Diagonal AND Check & Simple AND Check	Cyclic diagonal AND takes AND_AF, AND_EG, AND_'?'B and compares it to images H, C, D respectively. Similarly, AND_HA, AND_CE, AND_D'?' is compared to images F, G, B respectively and AND_FH, AND_GC, AND_BD is compared images A, E, '?'respectively. If this is close to 1, a raw score is assigned to the confidence array. Simple AND takes AND of adjacent images and compares it to the next image either horizonatlly or vertically. Again, if the Tversky's index for the compared images is above 0.98, the agent assigns a score accordinly to that partiular answer option.
	Bias/possible mistake : Comparion for diagonal AND is only done for Right Diagonal and would not work for problems with Left diagonal relationship.
Simple Invert XOR & Invert Offset XOR	This check only solves problem E-07 by comparing color inverted XOR (using PIL ImagesChops.Invert method) of adjacent images horizontally and vertically. An expression of how the agent computed Tversky's index is shown: **Tversky's Index(invert_XOR_AB, image C), Tversky's Index(invert_XOR_GH, image '?')** Invert Offset XOR only addresses problem E-04 and is very similar to the simple invert XOR. It starts of by offseting the images horizonally or vertically by certain no. of pixels, before applying color inversion and XOR to compute Tversky's index.
	Bias/possible mistake : Comparion only works horizontally and vertically, would fail for problems with diagonal relationships. For invert offset XOR, the offset was only applied from Left to Right horizontally and Top to Bottom vertically. The agent would fail to solve problems with offset relationships in the opposite direction.
Simple OR Check	Simple OR, just like Simple AND, takes OR of adjacent images and compares it to the next image either horizonatlly or vertically. Again, if the Tversky's index for the compared images is above 0.98, the agent assigns a score to that particular answer option.
	Bias/possible mistake : OR relationship is not implemented for diagonal relationships.
Cyclic Black Difference Check	This technique is specific for solving problem D-12 by With increase in no. of objects inside the image, formula shown below was implemented to check the black difference ratio and assign scores accordingly. **if (black_XB / 0.75)/(black_BD / 0.8)) > 0.95 and ((black_XB / 0.75)/(black_BD / 0.8)) < 1.05** If this 'IF statement' is true, a multiple of **((black_XB / 0.75)/(black_BD / 0.8))** is assigned as raw score for that option.
	Bias/possible mistake : This check was only implemented for Right diagonal relationships.
RMS Difference Check	Here the agent uses similarity measure as **RMSsimilarity = 1/(1 + RMS-diff)** adapted from[3]. The agent checks for the following expression. **abs((RMSsimilarityAC/RMSsimilarityBC) - (RMSsimilarityDF/RMSsimilarityEF))** If this is less than a certain threshold, the agent then assigns a certain raw score.
	Bias/possible mistake : Again, quite a few thresholds are checked before assigning the raw score. This might be feasible for this problem set but could deteriorate the performance for later problem sets.

3.4 Problem categorization and Weights

Recalling the flow diagram in Figure 3, the agent actually works through a problem on the basis of ease with which it can apply certain transform or method. The agent ranks identity check to be the easiest

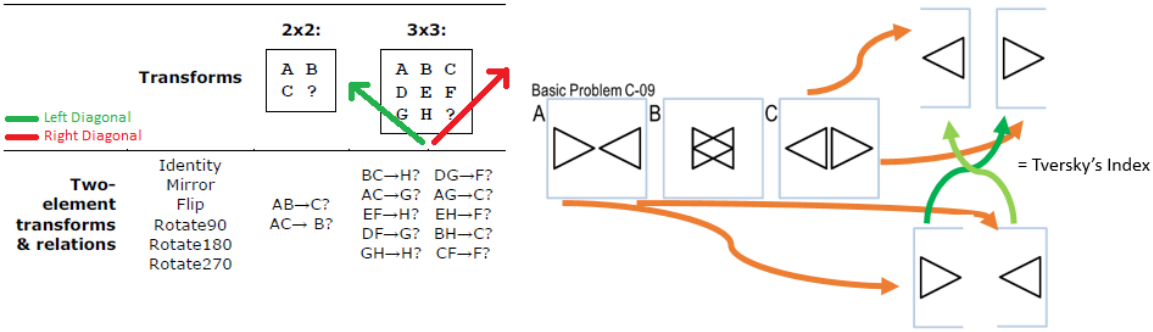

Figure 5: [LEFT]:Transforms and Matrix relationships implemented in the design, [RIGHT]: Problem C-09 Split

and RMS difference to be the hardest. Based on the ease of transformation, raw scores being assigned to the different answer options are a combination of similarity measure (Tversky's index), difference measures (Black pixel, RMS difference, etc) and similarity weights. Figure 6 [LEFT] shows a pseudo code of how the agent would actually go on assigning raw score when checking for identity transform. 'Identity CX' is the similarity between image C and a particular answer option and it is scaled up by a factor of 10 as the agent found this transformation to be the easiest.

```
identity_check():

    if identity_AB == identity_CX:

        raw_score[option] += 10 * identity_CX

    elif identity_AC == identity_BX:

        raw_score[option] += 10 * identity_BX
```

Similarity Weights 2 x 2

Weights	*Stages*	*Transformation*
10 points	1	Identity
8 points	2	Reflection
6 points	3	Rotation
4 points	4	XOR
2 points	5	RMS Difference
Impossible		

Similarity Weights 3 x 3

Weights	*Stages*	*Transformation*
15 points	1	Identity
14 points	2	Reflection
13 points	3	XOR
12 points	4	Black Difference
11 points	5	Split Compare
10 points	6	Enlarge Diagonal
9 points	7	Carve Diagonal
8 points	8	Diagonal AND
7 points	9	AND
6 points	10	Invert XOR
5 points	11	Simple OR
4 points	12	Invert offset XOR
3 points	13	Cyclic black diff
2 points	14	RMS Difference
Impossible		

Figure 6: [LEFT]: Detailed pseudo code example, [RIGHT]: Similarity weights and problem category

Figure 6 [RIGHT] shows how the agent is categorizing different problem based on ease of transformation and assigning extra weights to scale up the raw score. Similar to the pseudo code for identity check, the agent is multiplying the Tversky's index or absolute differences (as mentioned in the previous sections) it generates for each subsequent transformation checks (reflection, rotation, XOR, Black pixel, split comparison, etc) with an ease of transformation weight. The various stages are color coded and will be explained later on in the agent performance section. The 'Impossible' category is a problem where the agent is unable to apply any of the techniques successfully to get to a single answer with enough confidence. In this case it just returns the answer with shared confidence among the answer choices.

Bias/possible mistake : These weights were manually chosen for best overall results. They may be useful in this context but might hinder the performance of the agent for test problem sets.

4 Agent Performance

4.1 Accuracy

The agent correctly solved all 48 problems in basic problem sets B, C, D and E along with almost 6 correct problems in challenge problem sets B and C & almost 3 challenge problems in sets D and E. Figure 7 shows how the agent categorized different problems based on the stage of the problem in the process flow as per Figure 3.

Basic Problem Set B			Basic Problem Set C			Basic Problem Set D			Basic Problem Set E		
B-01	1	Correct	C-01	1	Correct	D-01	1	Correct	E-01	9	Correct
B-02	1	Correct	C-02	3	Correct	D-02	1	Correct	E-02	9	Correct
B-03	1	Correct	C-03	14	Correct	D-03	1	Correct	E-03	9	Correct
B-04	2	Correct	C-04	4	Correct	D-04	3	Correct	E-04	12	Correct
B-05	2	Correct	C-05	4	Correct	D-05	2	Correct	E-05	4	Correct
B-06	5	Correct	C-06	4	Correct	D-06	14	Correct	E-06	9	Correct
B-07	5	Correct	C-07	2	Correct	D-07	7	Correct	E-07	10	Correct
B-08	2	Correct	C-08	4	Correct	D-08	6	Correct	E-08	9	Correct
B-09	2	Correct	C-09	5	Correct	D-09	8	Correct	E-09	9	Correct
B-10	4	Correct	C-10	4	Correct	D-10	7	Correct	E-10	2	Correct
B-11	5	Correct	C-11	3	Correct	D-11	1	Correct	E-11	11	Correct
B-12	4	Correct	C-12	3	Correct	D-12	13	Correct	E-12	11	Correct

Figure 7: [Outer LEFT]: Set B, [Inner LEFT]: Set C, [Inner RIGHT]: Set D, [Outer RIGHT]: Set E

4.2 Efficiency

The agent solved the problems very efficiently in quick time. Figure 8 shows the average computation times for solving problems as the agent progresses through them. Most of the problems are solved within 5 s. On average, the total time agent took to address the basic B problems was 38.5 s, 107.5 s for basic C, 133.4 s for basic D and 224.8 s for solving basic E problems. The stages at which the problems are solved, as per Figure 7, coincide with the time it took to solve those particular problems.

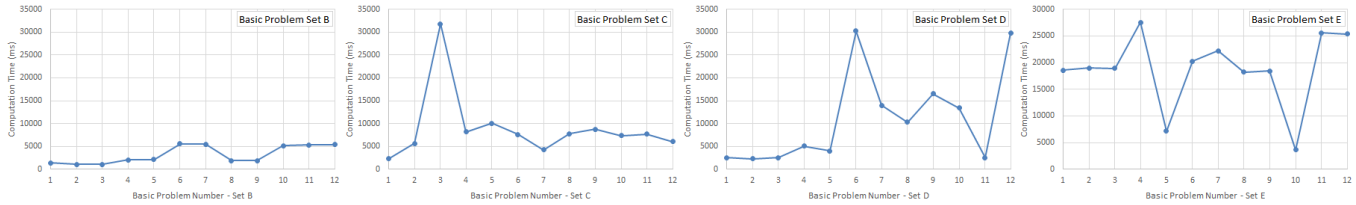

Figure 8: [Outer LEFT]: Set B, [Inner LEFT]: Set C, [Inner RIGHT]: Set D, [Outer RIGHT]: Set E

Possible delays : Agent may perform faster if Tversky index calculation algorithm or some of the difference check algorithms are more efficiently designed and implemented.

4.3 Generality

Although, the agent used limited number of methods to solve the problem, these methods (identity, reflection, rotation, XOR, AND, RMS difference, etc) were quite general as the agent was able to address all of the problems very well in the basic set and most of the problem in the challenge set. The assumption made in section 2.3 of this report paid off in the sense that the agent was able to generalize considerably well. This can be seen in the progressive solution matrix for each stage of the applied methods shown in Figure 9.

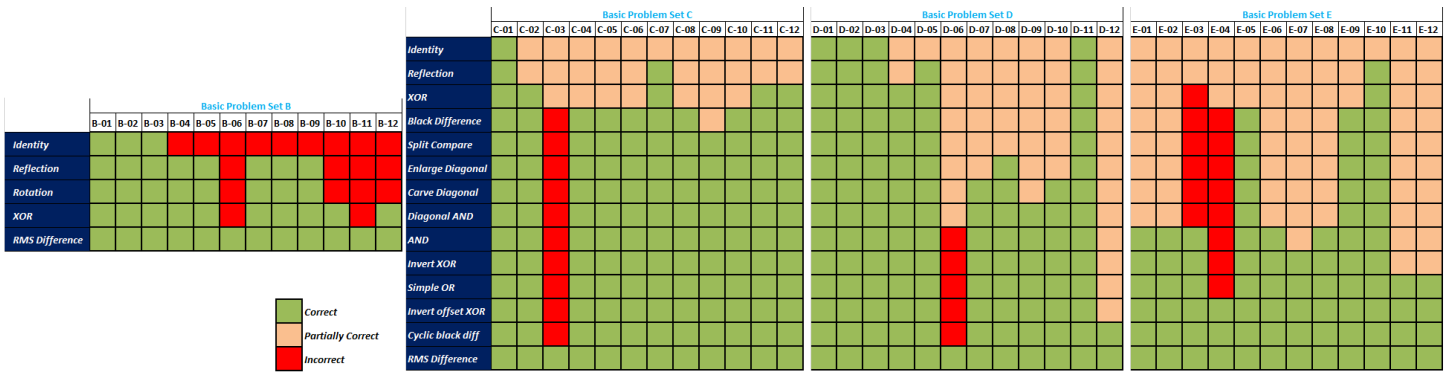

Figure 9: [Outer LEFT]: Set B, [Inner LEFT]: Set C, [Inner RIGHT]: Set D, [Outer RIGHT]: Set E

NOTE : The first thing that can be observed is rotation 2 x 2 does not really seem to help solve any additional basic problems for set B. For 3 x 3, the agent was designed in a such a way that if it is unable to solve a problem, it will assign uniform normalized score as evident by the partial correct answers for the initial stages.

5 Agent Design Imperfections

When applying transforms and calculating similarities and differences, there were 6 major imperfections that the agent design had. Already highlighted in the previous sections, these are as follows:

- **Preference :** Agent is designed to give preference to horizontal comparison over vertical comparison. This is the case for all the transformation techniques used. Again, the drawback of designing the agent this way is that vertical transform might be better or might provide a correct solution from human cognition point of view but the agent does not process it the same way. 'Basic Problem 05' is a very good example to illustrate this as shown in Figure 10 [LEFT]. It can clearly be seen that correct answer is option 4 which is just the vertical reflection of image B. As the agent prefers horizontal over vertical, it goes on to assign the raw score based on horizontal reflection rather than vertical. Although, the agent gets the answer correct but that is solely on the basis of similarity weight advantage (8 points) that the reflection has over rotation, XOR and RMS difference.

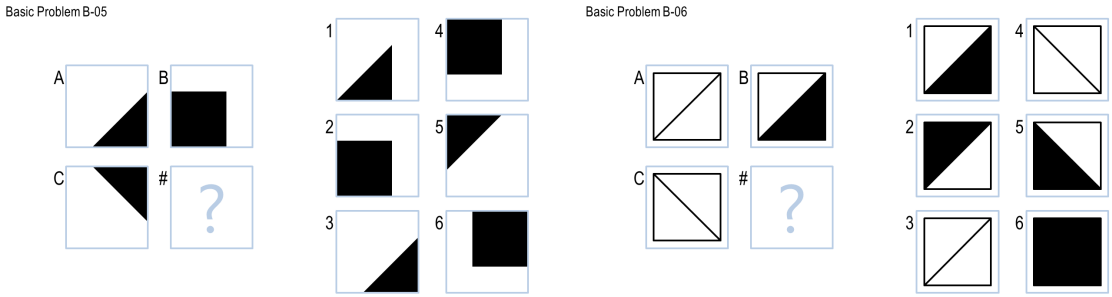

Figure 10: [LEFT]: Basic Problem 05, [RIGHT]: Basic Problem 06

- **Threshold :** The threshold values for all of the similarity and differences checks were set manually. Some tweaking was done in order to set the best possible values to satisfy all basic problems. Setting the threshold values this way is not a good practice and it might not work for more advanced problems which was the case when 2 x 2 agent created in project 1 was tested against the test problems as it was only able to solve 8 of the test problems.

- **Efficiency :** Although, the agent was quite efficient in running through all the problems but the design has definite room for improvement. The method that computes the Tversky index, computes the index by stripping down the data of the images into 1D arrays and then applies the formula. There can be other ways that this computation can be quickened in order to get efficient results.

- **Rotation Check :** The rotation check which was implemented as part of the design for 2 x 2 problems does not really contribute in solving any of the basic B problems. The reason for this could be the algorithmic implementation of the method. A good example to illustrate this is 'Basic Problem 06' as shown in Figure 10 [RIGHT], where the correct solution is just a 90 degree rotation of image B that is answer option 5, but the agent is unable to find the solution through rotation check (as evident from the progressive solution matrix in Figure 9) and ultimately relies on RMS difference to get to the correct solution.

- **RMS Check :** Although, the RMS check 3 x 3 solved basic problem C-03 but it was not an efficient solution as there was a lot of computation involved just to solve one problem. Another reason was, the RMS check was used as a last part of the process flow only because if it were used earlier in the flow, it would mess up all the weight distribution and generate incorrect results for some of the other problems in set C.

- **Diagonal Checks :** For problems in 3 x 3 sets, most of the methods do not use diagonal comparisons as part to of problem solving such as reflections 3 x 3, XOR 3 x 3 and split comparison 3 x 3. In fact split comparison 3 x 3 does not even cater for vertical comparisons. In contrast to this, for the cyclic black difference check, only diagonal relationship check was implemented.

There are other smaller inconsistencies also that the agent design has but the ones mentioned above can be of greater concern when solving difficult problems.

6 Design Rectifications/Improvements

With availability of unlimited time and resources, there are few amendments that can be done to the design in order to address the imperfections mentioned above and achieve more accurate results even more efficiently.

- **Preference :** A possible solution for the horizontal reflection over vertical reflection preference can be to check for colored pixel count at specific indexes before choosing which form of reflection to apply, i.e. first compare the colored pixel count for image A - B at each index to colored pixel count for image C - '?' and then do the same for image A - C and compare it to colored pixel count for image B - '?'. If pixel count for image A - B and C - '?' is more comparable the agent should choose horizontal reflection or otherwise if A - C and B - '?' is more comparable then agent should choose vertical reflection. This is just a general idea of how to check which reflection to apply and the actual implementation might differ. The idea is still in development and I would try to implement it as part of project 3. One other way this can be addressed is not to assign raw scores right away, i.e. while applying transformations, just store the calculated similarity measure in a variable and at the end just assign the max measure to the raw score. This imperfection is carried on from project 1 and was not addressed simply because of lack of time.

- **Threshold :** Instead of choosing a single threshold that might solve basic set of problems but not work for difficult ones it will be better to loop through an array of discrete threshold values for each problem and settle with the one that give the best result. This problems has also not been addressed since project 1 as focus was solely in solving problems from problem set C. Instead of looping through multiple values at run time, which might take a lot of computation time, I am thinking of only choosing 2 values for thresholds and then choosing the raw score that maximizes the similarity or difference measure.

- **Efficiency :** As of now the Tversky's index is being computed by comparing 2 linear 1D arrays and applying the formula shown in Figure 2. Another way to implement this formula can be to compare pixels of the images in a nested for loop using 2D image arrays. This would definitely be a quicker way to compute the similarity measure. As Tversky index is during the application of every transform, this can really speedup the computation time and will greatly help with difficult problems. Again, this issue is causing further delay when solving 3 x 3 problems as well. Instead of using getdata() method from PIL python library to calculate Tversky's index, load() method can also be used and then the pixels can be compared in form of a 2D array directly without a excessive amount of delay.

- **Rotation Check :** Rotation comparison implementation only considers that the images can be rotated by 90, 180 or 270 degrees. May be adding more degrees like 45, 135, 225 and 315 would help in applying rotation to difficult problems in the future. Agent design is only comparing the maximum of similarity measure up till now, if given unlimited time the rotation check method can be designed to check similarity for each degree of rotation. However, in the little time that I had while designing agent for project 2, I did made an unsuccessful attempt to come up with a possible solution to this problem as I am sure this is the reason that the agent was also unable to solve some of the test B problems correctly.

- **RMS Check :** I would probably try to come up with a better method to solve problem C-03 in project 3 because of the inefficiency of this method. May be I will modify XOR 3 x 3 or Black difference 3 x 3 methods to address this problem early on rather that trying to solve it at the end.

- **Diagonal Checks :** If given unlimited time, I would try and add the capability of checking for all horizontal, vertical and diagonal relationships for each method used to solve the problems.

7 Human Cognition and Artificial Intelligence

The agent was designed in order to naturally simulate how humans would attempt to solve these RPM problems. Normally, humans breakup problems into smaller sub-problems and then apply rational thinking to solve those sub-problems, ultimately coming up with a solution to the bigger problem.

While working on IQ problems, specifically RPMs, humans are able to solve the easier problems in almost negligible time and tend to spend more time as the problems get harder. This is exactly what this AI agent is doing as it is evident by the computation time curves in Figure 11, the agent is quick in responding to problems where only identity check is necessary to obtain the correct answer, i.e. basic problems 1, 2 and 3 for set B and problem 1 for set C. But then it spends more time as the problems get harder. Although, the agent is not as quick at responding as a human, it is still closely following the computation time pattern to that of a human while solving these RPM problems.

Another aspect where the agent formulates a methodology of what human would do while solving the RPM problems is to break it into sub-problems as discussed earlier. By breaking the problem into smaller manageable categories such as shown in Figure 10 as easy, moderate, hard, etc, the agent then (to the best of its ability) tries to address each problem on its merit. This is primarily what a human would also do in a situation where he/she faces a difficult problem, i.e. apply 'divide and conquer' rule.

As there was a choice to design the agent based on verbal or visual approach, by designing the agent based on visual representation, it more closely relates to how humans would extract information from the images provided in the RPM problems. In most cases humans would rely on image relationships which they are able to see visually rather than noting down image properties verbally to solve the problems

8 Conclusion

This project was a great way to understand different aspects of artificial intelligence and its application to solve RPM problems. Designing the agent based on visual representation also helped me further explore python PIL library. Overall the project was really interesting and I enjoyed applying different geometrical and logical transforms to images in order to answer the problems, specially when solving problem C-09 and some of the difficult problems in set D and E.

References

[1] **Kunda, M., McGreggor, K., & Goel, A. (2010)** *Taking a Look (Literally!) at the Raven's Intelligence Test: Two Visual Solution Strategies.*

[2] **Kunda, M., McGreggor, K., & Goel, A. K. (2011)** *Two visual strategies for solving the Raven's Progressive Matrices intelligence test.*

[3] **Kunda, M., McGreggor, K., & Goel, A. K. (2012)** *Reasoning on the Raven's Advanced Progressive Matrices Test with Iconic Visual Representations.*

[4] **Kunda, M., McGreggor, K., & Goel, A. K.** *Addressing the Raven's Progressive Matrices Test of "General" Intelligence*

[5] **Amos Tversky (July 1977)** *Features of Similarity*

[6] **Similarity Measures** *https://docs.eyesopen.com/toolkits/python/graphsimtk/measure.html*

[7] **Fredrik Lundh (March 17, 1997)** *http://effbot.org/zone/pil-comparing-images.htm*