Weighted Voting on Raven's Progressive Matrices Tests

David Armes
School of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332, USA
darmes3@gatech.edu

Abstract

Using purely visual representations of the Raven's Progressive Matrices Tests, this paper proposes a distributed system of analysis and weighted voting to determine both answer and confidence. The details and performance of the proposed system are shown to be effective for over 75% of tested problems.

Introduction

The Raven's Progressive Matrices (hereafter RPM) is a common intelligence test that relies on visual reasoning and is therefore language independent. The tests typically consist of a 3x3 matrix of images with a single missing image, which the test taker is supposed to supply from a list of multiple choice images.

An example of an RPM test question is shown in Figure 1. Each question was presented to the AI as a series of individual images, that is as 16 distinct images. With each image labelled as to whether it was part of the problem – and what spot it belonged to (A, B, etc.) – or as a potential solution. The agent returned either a numerical guess corresponding to the image it thought was the answer or a null value if it was uncertain.

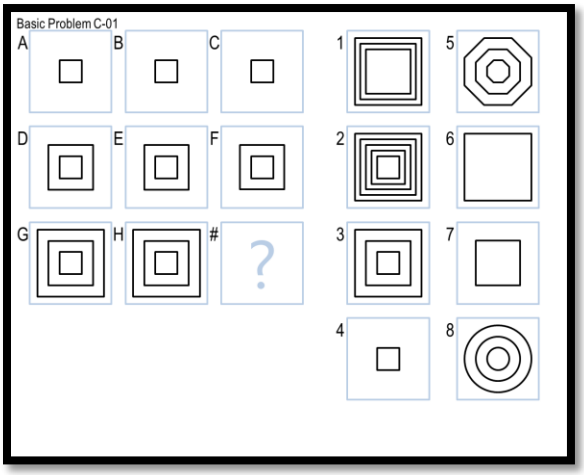


Figure 1, Example RPM Test Item

Operators

In order to keep the agent as computationally efficient as possible, the agent was given access to a limited set of image operators. The list of operators is as follows along with their associated shorthand notation:

- No transform (r0)
- Rotation 90° (r90)
- Rotation 180° (r180)
- Rotation 270° (r270)
- Mirror X (mX)
- Mirror Y (mY)
- Intersection / Lighten (i)
- Union / Darken (u)
- Relative Complement (rel)
- Difference (diff)

The first six operations are performed on a single image; examples of the operations are shown in Figure 2. The last four operators must be performed on pairs of images; examples of the operations are shown in Figure 3. Note that while the relative complement operator is order dependent, the other operators are order independent. A note on nomenclature: assuming a given image A and B, then iAB is the intersection of images A and B. Likewise, you might also have u(r90A)B which is the union of image A rotated 90 degrees and image B. Finally, because the most common operation is difference, the 'diff' prefix is often dropped. Therefore, 'AB' is equivalent to 'diffAB'.

Agent Design Overview

The AI agent uses two main systems, dubbed the "voting" and "ranking" systems. The voting systems consists of multiple "voters". A voter looks to see if a particular feature is present in a problem. If it isn't present, then it abstains from voting. But if

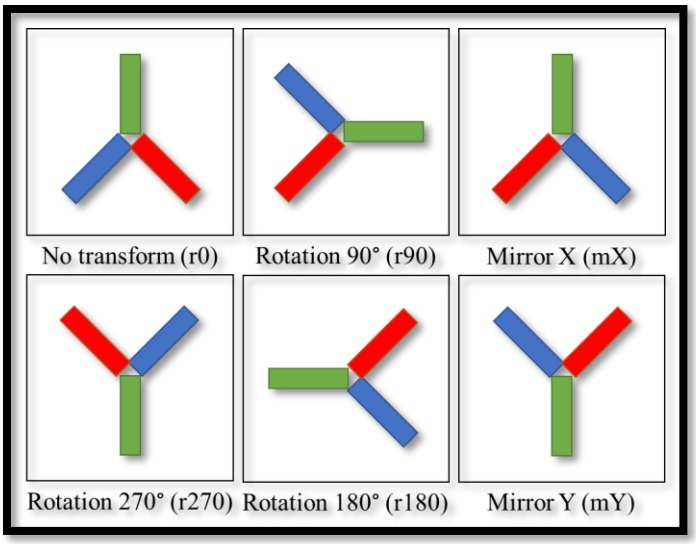


Figure 2, Single Image Operations Examples

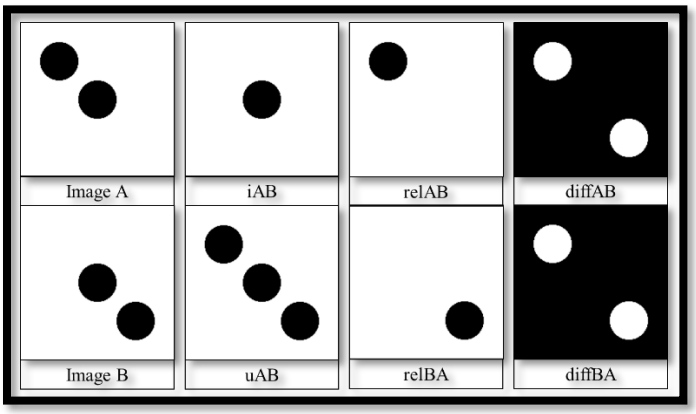


Figure 3, Dual Image Operations Examples

```
def voter_generic(problem):
    if feature_is_present():
        create_guess()
    else:
        abstain()
```

Figure 4, Pseudocode for a Generic Voter

```
def ranker_generic(answers):
    guess = get_guess()
    ranked_list = [ ]
    for answer in answers:
        score = score_answer(answer, guess)
        ranked_list.append( (score, answer) )
    return ranked_list.sort()
```

Figure 5, Pseudocode for a Generic Ranker

it is present, then it generates a guess about what the answer will look like based on that feature.

After all voters have cast their vote, the ranking systems takes over. A ranker looks to see if its companion voter created a guess or abstained. If a guess was made, the ranker scores each potential answer by how well it matches the guess, and returns an ordered list of the answers arranged by their scores. These ordered lists are then combined to produce a single, final answer. A graphical depiction of the process is shown in Figure 6.

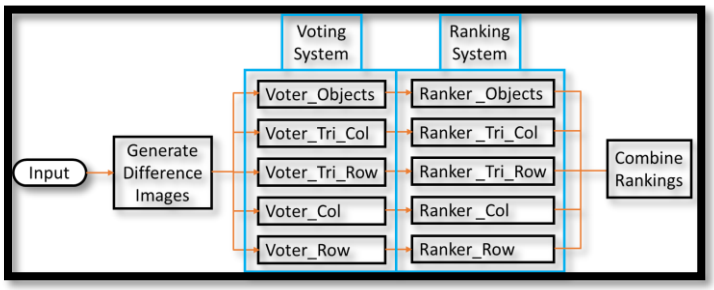


Figure 6, Overview of Agent Design

Voters

As a precursor step, ten difference images are generated from the problem images (A-H). This creates a fixed set of 18 images

Figure 7 for the voters to work with although some of the voters may create additional images as required. The different voters are listed below:

- Voter Row / Voter Column
- Voter Tri Row / Voter Tri Column
- Voter Objects
- Voter Combination
- Voter Union Row / Voter Union Column
- Voter Inverted Row / Voter Inverted Column
- Voter Intersection Row / Voter Intersection Column

Each voter is explained separately below along with an example. For the pairs of Row/Column voters, only one will be explained as the other works identically but uses rows or columns respectively.

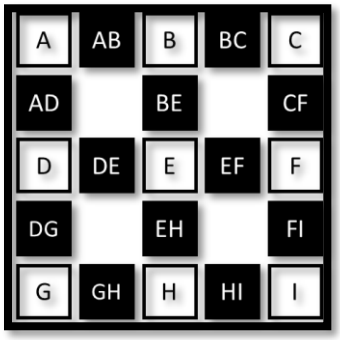
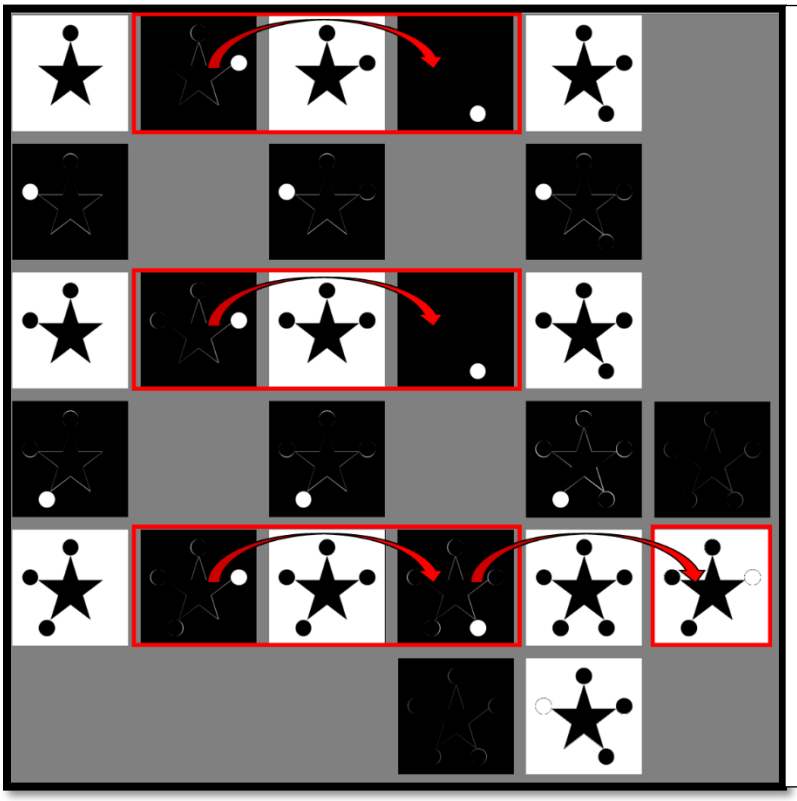


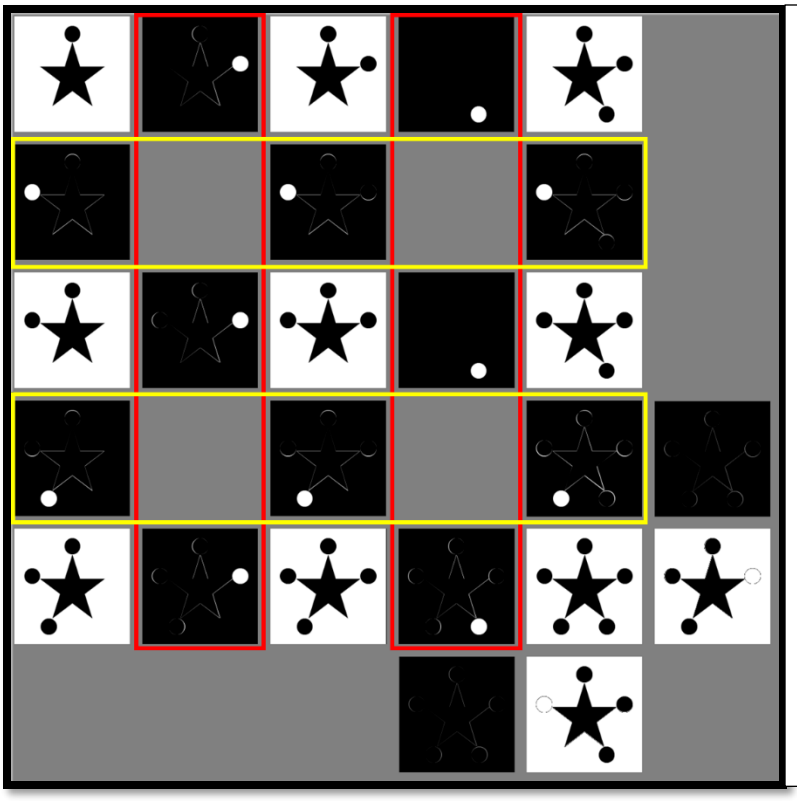
Figure 7, 3x3 Image Nomenclature



1. The Row voter looks for similarity between the pairs AB-BC and DE-EF. It can use rotational and mirror transforms to achieve a better match. Specifically, it can use 0, 90, 180, and 270 degree rotations and mirrors across the x and y planes.

2. Here, the voter has decided that no transform is necessary to get a good fit. It then applies the transform to GH and then performs a difference on the transformed GH and H to generate the guess image. Note that this guess is the same as image G, and not actually correct.

Figure 8, Voter Row for Problem C-05



1. The Tri_Row and Tri_Col voters look for similar rows/columns. If the AB-DE-GH images are similar, then BC-EF-HI should be similar as well.

2. The mostly black images next to FI and HI are the comparison images generated by these voters. These show that this answer is a good fit.

3. Like all voters, this similarity may not hold true from problem to problem, so a threshold value is used to determine whether the row or col is actually 'similar'.

Figure 9, Voter Tri Row for Problem C-05

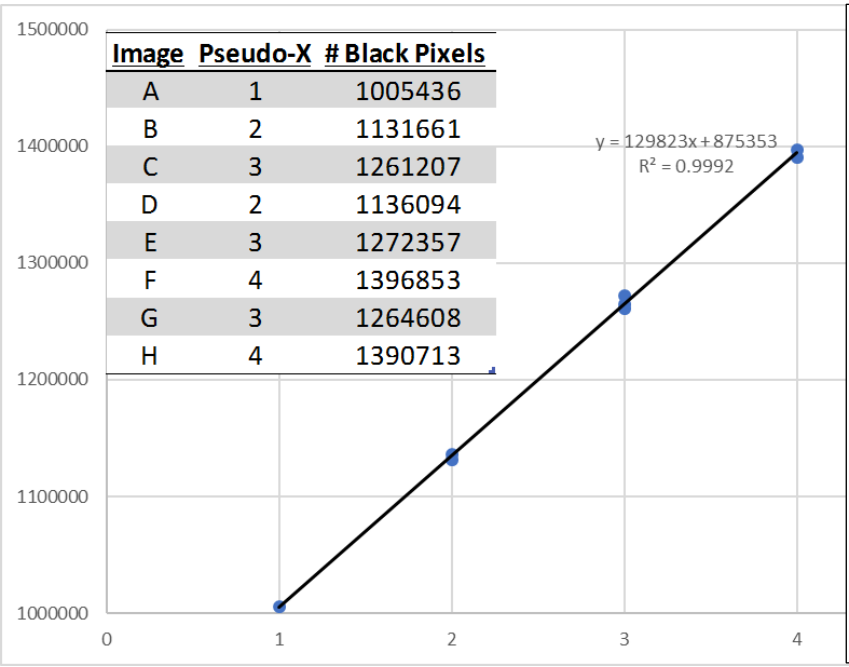


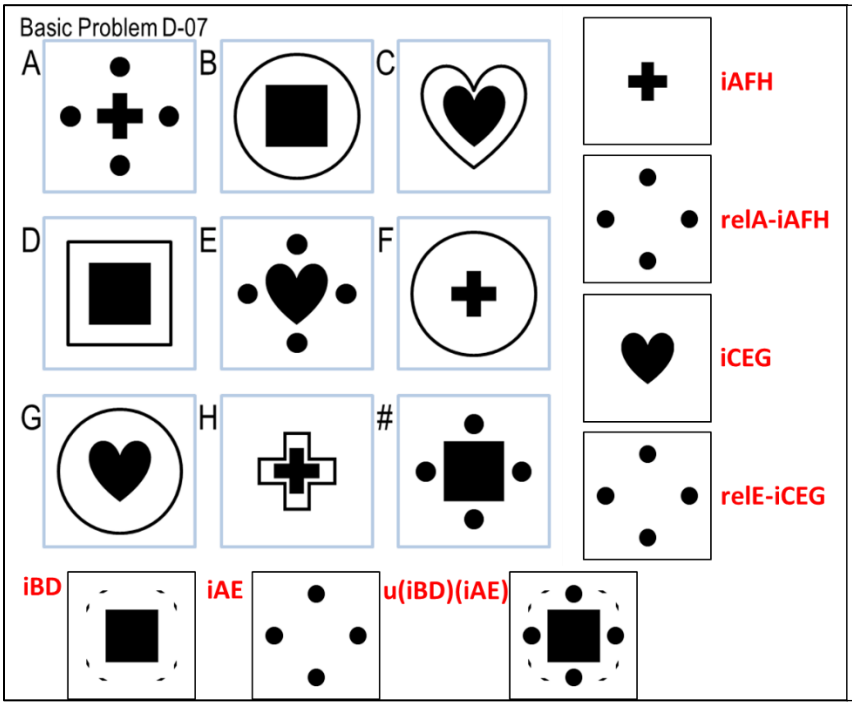
Image	Pseudo-X	# Black Pixels
A	1	1005436
B	2	1131661
C	3	1261207
D	2	1136094
E	3	1272357
F	4	1396853
G	3	1264608
H	4	1390713

1. The Object voter looks for trends in the number of 'objects' in the problem.

2. Each image is assigned an 'x' value based on how far it is from image A. The number of black pixels is then calculated for each image.

3. Using this data, a linear regression analysis is performed. If there's a good fit, then the voter calculates a value for a pseudo-x of 5. In this instance, it calculated a value of 1,524,467 with the correct answer having a value of 1,523,780 or a 1.2% difference.

Figure 10, Voter Objects for Problem C-05

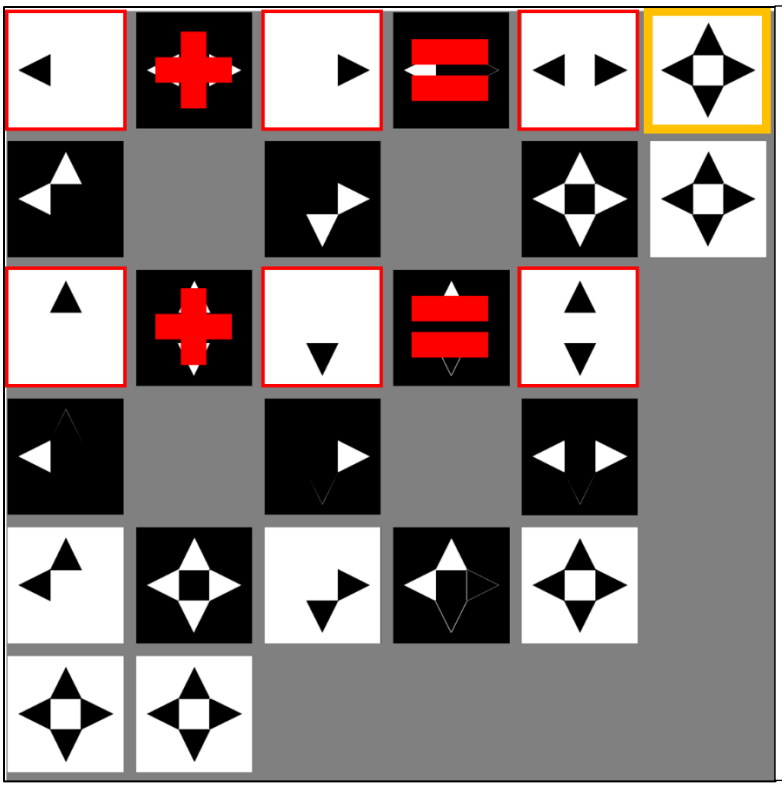


1. The Voter Combination looks for when features from two separate sources are combined to produce a result.

2. The intersections iAFH and iCEG are created. This yields the common shape among them. Then the relA-iAFH and relE-iCEG are created. This creates the feature common to A and E. These images are compared to see if they match.

3. If they do match, a guess is created by taking iBD and iAE and creating their union u(iBD)(iAE). This guess is very close to but not quite the same as the correct answer.

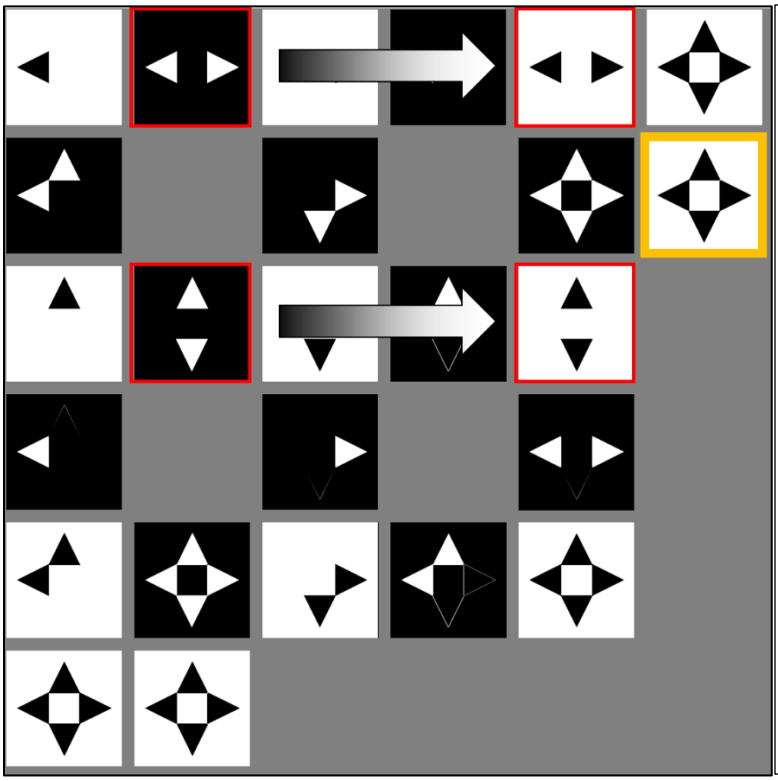
Figure 11, Voter Combination for Problem D-07



1. The Voter Union Row checks if the last image in a row is the union of the first two, i.e. (uAB=C) & (uDE=F)

2. If they are, it generates a guess by creating uGH (shown in orange).

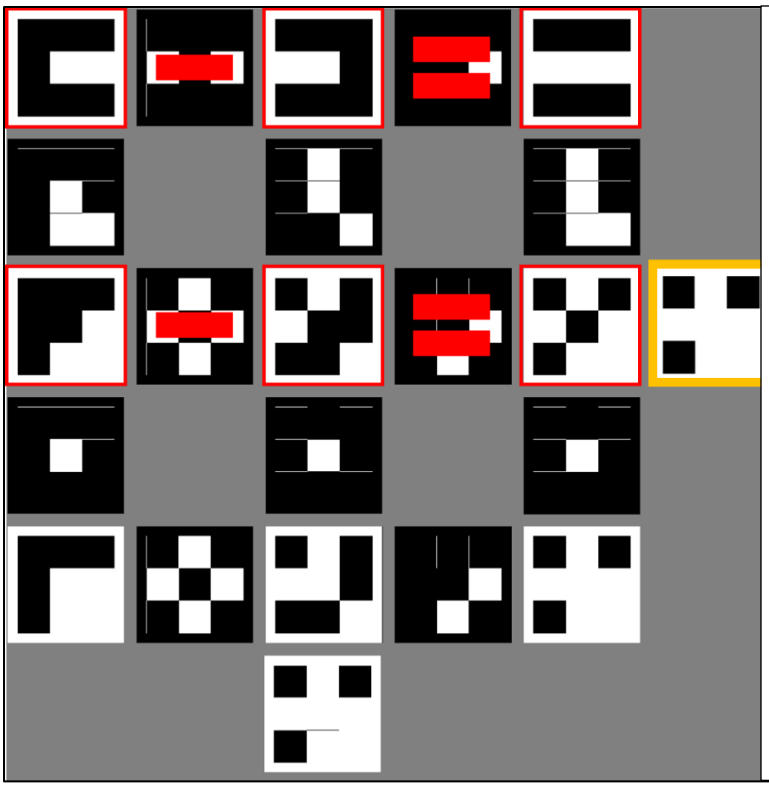
Figure 12, Voter Union Row for Problem E-01



1. The Voter Inverted Row checks if the last image in a row is the inversion of the difference of the first two, i.e. (AB=C) & (DE=F)

2. If they are, it generates a guess by inverting GH (shown in orange).

Figure 13, Voter Inverted Row for Problem E-01



1. The Voter Intersection Row checks if the last image in a row is the intersection of the first two, i.e. (iAB=C) & (iDE=F)

2. If they are, it generates a guess by creating iGH (shown in orange).

Figure 14, Voter Intersection Row for Problem E-11

Ranking	1st	2nd	3rd	4th	5th	6th	7th	8th
Weights	10	8	4	2	-2	-4	-8	-10
ObjectVote	'3'	'6'	'8'	'5'	'1'	'4'	'7'	'2'
TriRowVote	'3'	'6'	'8'	'5'	'1'	'4'	'7'	'2'
TriColVote	'3'	'6'	'5'	'8'	'1'	'4'	'7'	'2'
RowVote	'8'	'6'	'1'	'3'	'4'	'7'	'5'	'2'
ColVote	'5'	'1'	'3'	'7'	'6'	'4'	'8'	'2'
Answers	('3' 36)	('6' 30)	('8' 12)	('5' 10)	('1' 6)	('4' -18)	('7' -26)	('2' -50)

Figure 15, Ranking System Orderings

Ranking System

Once all the voters have made their inputs, it's time for the ranking system to sort through the answers and determine what matches. Where possible, it uses the image difference function to compute similarity. An example ordering based on an actual problem is shown in Figure 15. Note that not all voters are present indicating that some of them abstained.

As can be seen the Object, TriRow, and TriCol voters all prefer answer number 3 the most, but those three and the Row voter also prefer answer 6 quite heavily as well. Rather than using a 'winner-take-all' approach, the ranking system uses weights to determine how each voter's contribution impacts total choice. So, answer 3 came in 1^{st}, 1^{st}, 1^{st}, 4^{th}, and 3^{rd}, therefore it receives a total rank of (10+10+10+2+4) = 36. By contrast, although answer 6 was heavily preferred by 4 of the 5 voters, it gets penalized by the Col voter and has a total rank of only 30. Thus, the ranking system declares answer 3 as its guess, which is in fact correct.

Limitations

The agent is severely limited in cases where none of the voters produce guesses. If none of the voters can find any matches, then no guesses are generated, and the final rankings are all zero. However, this lack of information is still useful in that the agent knows that it doesn't know anything and so it can skip the problem rather than return an incorrect guess.

A further limitation is that as new voters are added they can potentially conflict with existing voters and latch on to incorrect solutions. This caused the agent's performance to drop on previously solved problems when re-testing.

To correct this, the values used for thresholding had be individually modified to prevent conflict. Each voter contains at least one threshold value that is uses to determine whether a feature is present. If the threshold is set too loosely, then every problem qualifies as having the feature and the voter generates incorrect guesses. Conversely, if the threshold is set too tightly, then no problems show as having the feature and the voter serves no purpose. As each voter has at least 1 threshold value, then there at least 12 values to tweak to improve classification hence this can be thought of as a 12-dimensional decision boundary problem. In future research, this could be an interesting area of study, but for now a manual inspection and setting of the values was sufficient.

Results

To test the performance of the agent, a series of problem sets were used with both training and test sets. The sets progressed in difficulty from set B to E. Each 'basic' set contained 12 problems which the agent could work and solve, and which the researcher had access to study. Each 'test' set was kept blind from the researcher, and only used to evaluate the agent's performance.

Overall, the agent performed well, solving 83% (10/12) in all of the Basic sets and nearly as well at the Test sets. A separate metric though is the ratio of incorrect to skipped problems. Of the non-correct problems, approximately 60% (11/19) are incorrect guesses by the agent rather than skips; by comparison, the agent gets 80% of all the problems correct. This suggests that there is more room for improvement in the agent's meta-cognition. Put another way, more often the agent is confident in a wrong decision than willing to admit defeat.

This is due to the agent's overall design. Voters make decision based on simple yes/no thresholds, and thus have no way of communicating their confidence in their vote. Similarly, the rankers give each vote the same weight without considering how well it really applies. Lastly, the decision to skip is made only when there is no consensus among the voters and therefore never considers that they could all be wrong.

Problem Set	Correct	Incorrect	Skipped
Basic B	10	2	0
Test B	10	2	0
Basic C	10	1	1
Test C	10	1	1
Basic D	10	1	1
Test D	9	1	2
Basic E	10	0	2
Test E	8	3	1
TOTAL	77	11	8

Figure 16, Agent's Performance on the Problem Sets.

Conclusion

In summary, the agent's performance in terms of correctness is only average. However it's true strength lies in the simplicity of its methods. It uses no high level image recognition techniques, and yet is still able to complete the test with performance on par with an average human. This showcases the power of even relatively 'dumb' image processing methods to perform higher cognitive functions.