

Project 2

# **Solving 3x3 RPM using Verbal and/or Visual Representations**

---

Aayush Kumar

27th September, 2017



## Introduction

Stemming from project 1, I was able to leverage several of the design decisions I made early on with my implementation of Semantic Networks.

Without further ado, here is a technical approach to solving Raven's Progressive Matrices:

1. Project 1 Overview
2. Changes in problem set & Agent's reasoning
3. Agent capabilities
4. Agent limitations
5. Agent performance
6. Agent's relation to KBAI

## Project 1 Overview

*Please provide a short overview of your approach in Project 1.*

As mentioned beforehand, my agent emphasizes its knowledge processing using the verbal/textual cues provided by the problem statement. I had some basic image processing, but was more interested in testing the capabilities of the Semantic Network paradigm, thus I designed a class hierarchy that encapsulated all properties of a Semantic Network as well as finding differences between two semantic networks.

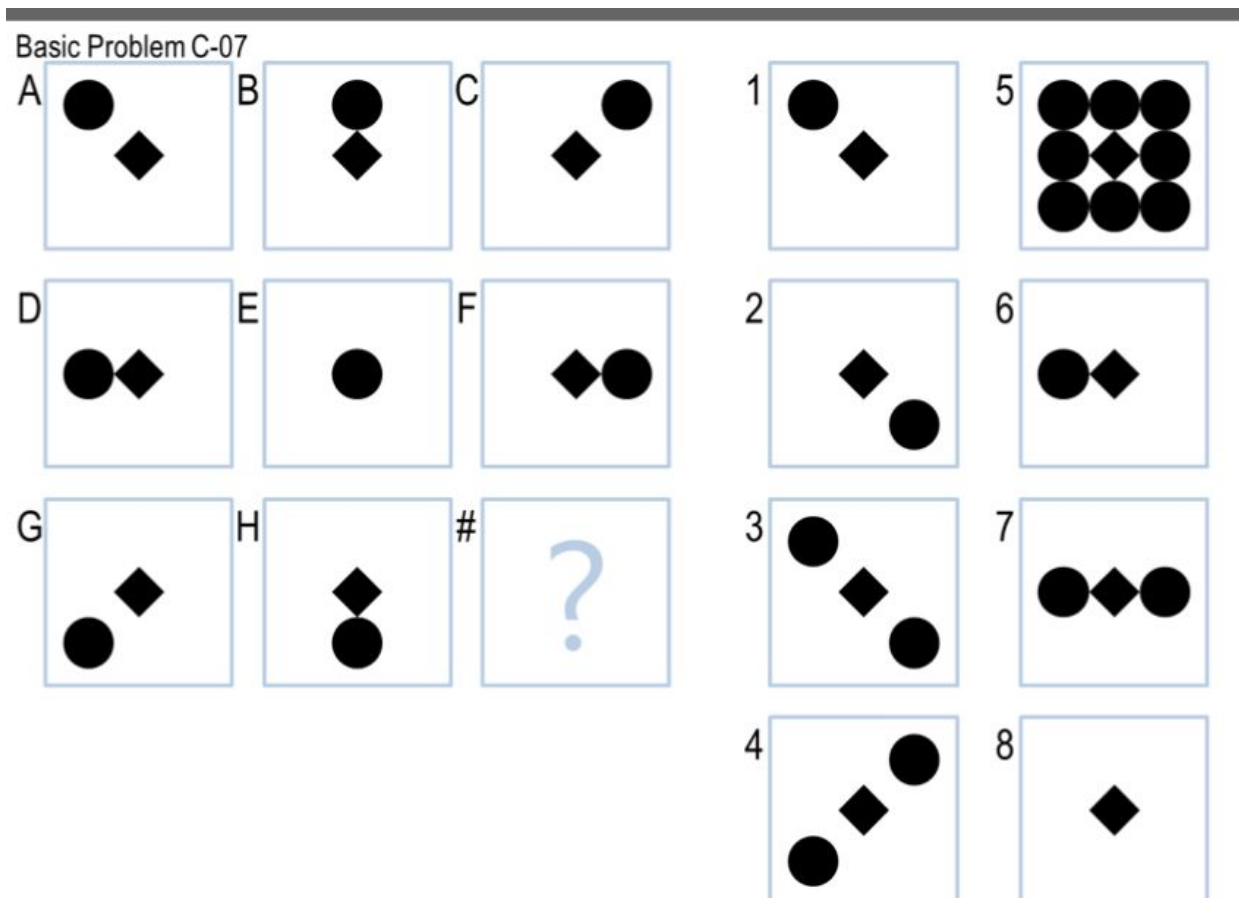
## Changes in problem set & Agent's reasoning

*How was the problem set for Project 2 different from Project 1? What other challenges did you face? Did you change your input representation to incorporate these changes? If yes, how did you make sure that backward compatibility still exists?*

- added object count interpolation
- better object matching
  - matching precedence give better overall optimal soln
  - implemented a vectorized notation of objects as well as nearest neighbor algorithm to derive objects that match each other in converted, numerical attributes
  - look to object categories that haven't been exhausted for matching remaining objects

Previously we encountered 2x2 Progressive matrices, which proved to be relatively challenging but still reasonable enough to translate into the analogy of A corresponds to B as C corresponds to # and A corresponds to C as B corresponds to #. However, with the introduction of 3x3 Raven's Progressive Matrices, we encounter a host of new problems- the largest, overarching of which is how to account for all three rows and three columns in predicting #. I luckily did not have to completely overhaul my input representation for these changes, but I did feel the need to

modify it slightly and make it more robust for the next phase in Project 3. My Agent's primary source of reasoning relies on the manipulation of the knowledge representation it generates from the problems given. Thus, getting that initial representation right was crucial, and more specifically perfecting my object matching. One problem I noticed was that my approach of using a source figure to derive pairings between it and a destination figure were hindered by the situation where my source figure has very few objects in it. Take the following example:



Basic Problem C-07.PNG

In trying to compute the object pairings for H, we might use E as a reference figure. That is, we match H's figures with that of E's to the best of our ability, and then if we still have remaining figures in H then we assume that a new object has been introduced into the problem. In this case, we see that the circle matches between both images, but there is also a diamond in H so it seems like it's a brand new object with no pairing. However, in reality we have seen this object before as it appears in every figure in the problem statement except for E.

To combat this issue, I changed my algorithm to do the following

```

// match objects from Figure A to Figure B
matchObjects(FigA, FigB)

Let FigSrc = Figure with less objects
Let FigDst = Figure with more objects

for objSrc in FigSrc:
    match with closest objDst in FigDst
    remove objDst from future candidate matches

for objDst in FigDst:
    match any remaining objects to objects that we haven't
considered but have seen when computing previous matchObjects(_,_)

if FigDst is FigSrc:
    invert the matchDictionary and swap keys and values

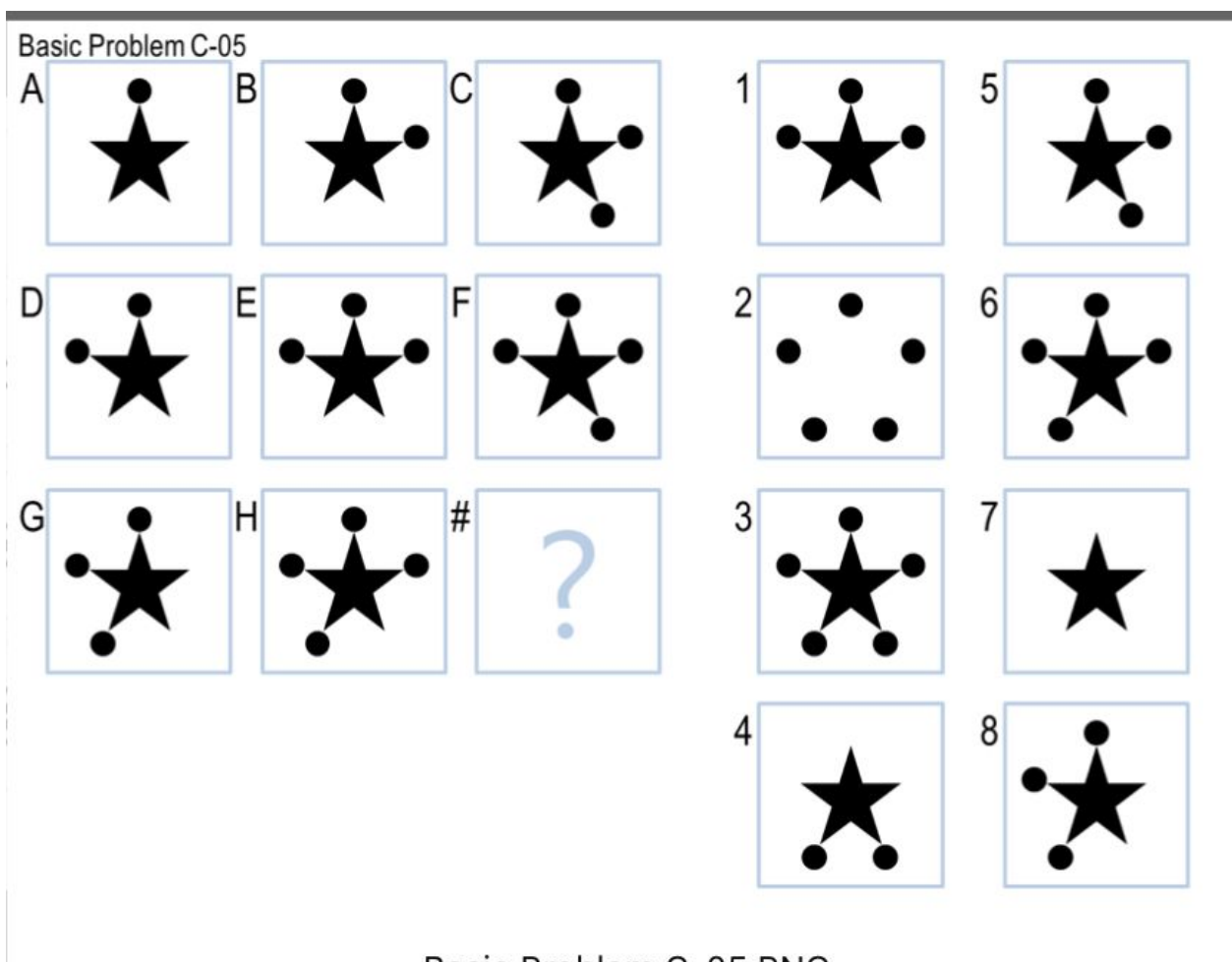
```

By explicitly matching from the smaller number of objects to the larger number of objects, I guarantee that the matches I find will be optimal and any remaining objects can be handled in one of two ways. Either I am able to find an object category that I have created in the past but has currently disappeared, or I can create a new object category. (See [extractAliasPairing.py](#) in [ObjectMatching.py](#))

In regards to future proofing this process, I implemented a means by which to create vectorized representations of individual objects. This way, I can use the nearest neighbor algorithm I wrote to derive object pairings based on individual attributes represented as numbers. This will come in handy when we look at less intuitive patterns- ie: a shape that increases by one side each column over or row downward. In this case, I store the sides as a number too in my vectorized representation of the object.

## Object Count Interpolation

One thing I immediately noticed about solving 3x3 Raven's Progressive Matrices was the patterns in how many times objects individually or collectively appeared in each row or column. These patterns could very well be used to extrapolate equations of curves and lines, thereby enabling me to derive the corresponding count(s) of objects in my answer. Take the following image for example:



Here, let's represent this problem numerically by how many objects we see in each figure:

2	3	4
3	4	5
4	5	?

Using each column's index and row's index, we can extrapolate the coefficients of quadratic equations for the top two rows and similarly for the left two columns. In this case, we have:

2	3	4	[0, 1, 2]
3	4	5	[0, 1, 3]
4	5	?	-
[0, 1, 2]	[0, 1, 3]	-	-

Putting these coefficients in a table gives us the ability to interpolate the coefficients for the last row and column:

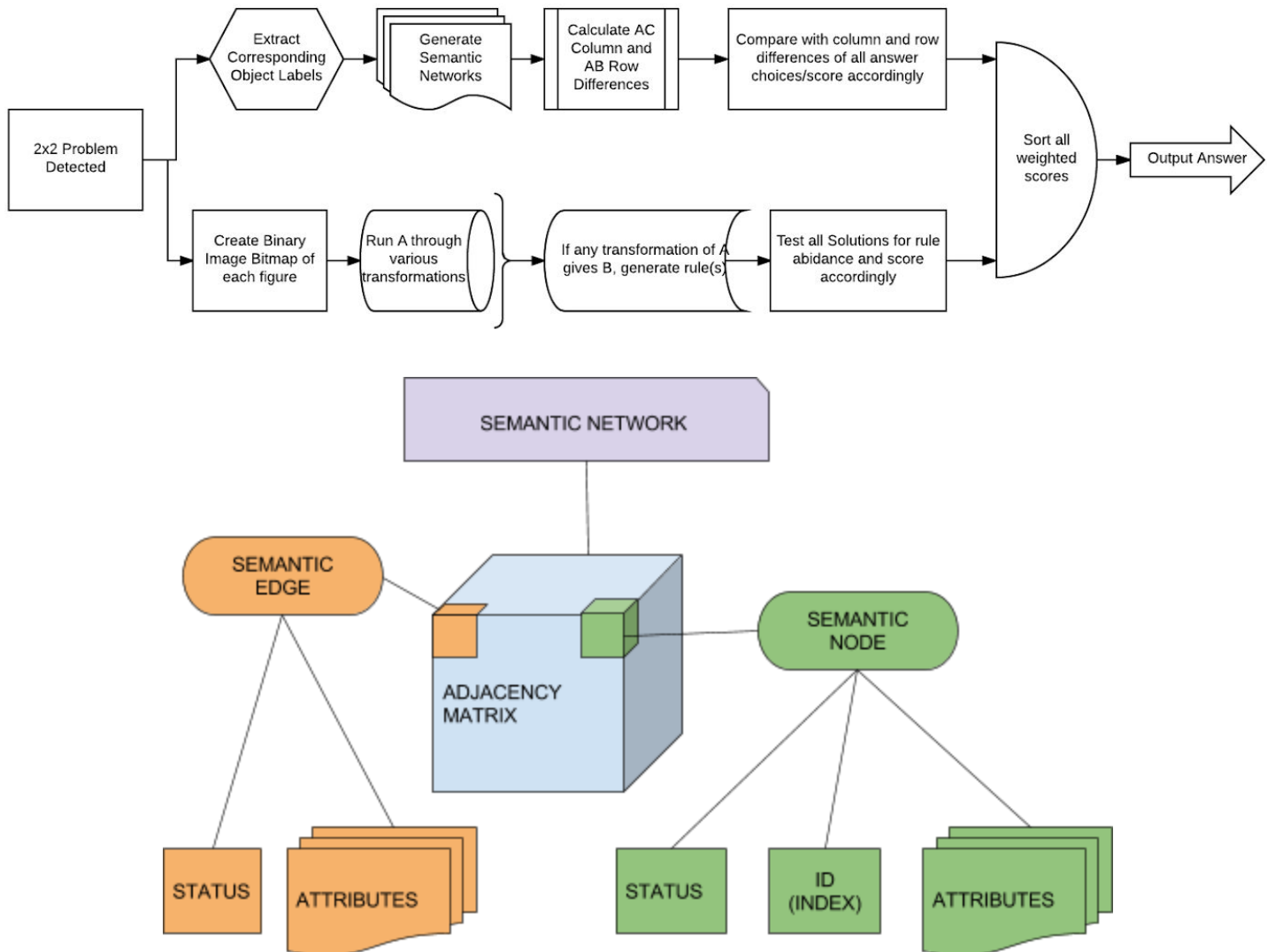
row	ax <sup>2</sup>	bx	c	col	ax <sup>2</sup>	bx	c
0	0	1	2	0	0	1	2
1	0	1	3	1	0	1	3
2	0	1	4	2	0	1	4

Expected Number of Objects in Answer via Row =  $0 \cdot 2^2 + 1 \cdot 2 + 4 = 6$

Expected Number of Objects in Answer via Col =  $0 \cdot 2^2 + 1 \cdot 2 + 4 = 6$

## Agent capabilities

*Provide an overview of the design of your agent*





## Agent limitations

*Have you rectified your previous version's mistakes? Are there new mistakes that your agent now makes in the new problem set? Could these mistakes be resolved within your agent's current approach, or are they fundamental problems with the way your agent approaches these problems?*

- was able to account for quantitative analysis across 3x3 grid, but most of the qualitative analysis still only resides in the bottom right 2x2 grid, need to improve this somehow

Perhaps one of the largest challenges of this project was tackling the qualitative analysis across both the 2x2 and 3x3 boards alike. Previously, I felt that I reasonably captured that information in my Semantic Network Adjacency Matrix and could be used as a primary knowledge base for understanding and solving any problem. Thus, my agent emphasizes this methodology, and I continued to hone this Semantic Reasoning even further this time around. However, I am starting to feel that further and better qualitative analysis will come from more advanced image processing techniques beyond the scope of this class despite the suggestions of not needing to use Computer Vision. This was largely the influence of my Nearest Neighbor approach for down the road, as it's a technique that's often used with a bag of sift features. (CS 4476 James Hays, Georgia Tech/Udacity) In this approach, descriptors of interest points are calculated and taken into consideration when doing feature matching. Down the road, I would like to leverage this idea further and hopefully improve drastically on qualitative reasoning.

## Agent performance

*Using the new changes what was the performance of the agent? Did you use the same evaluation metrics as earlier? What were the new ones, if any? Did your agent generalise well enough from earlier without the new changes?*

My agent performs quite fast for solving each question. Relative to last Project's runtimes, my agent ran an average of only 6.4% slower over 20 runs through Problem Set B and C. In addition, changing the weights for my vector representations and modifying the similarity scoring algorithms as follows proved to give more optimal solutions. All terms are explained in depth in the docs of my code!:

Same Status	+17.5
-------------	-------

Same Attribute Value	+2
Same Attribute, different Value	-0.15
Uncommon Attribute	-0.15
None Comparison	-0.2

```
weights = np.asarray([

    0.500, # width

    0.500, # height

    0.200, # fill1

    0.200, # fill2

    0.200, # fill3

    0.200, # fill4

    1.000, # shapeID

    0.500, # shapeSides

    0.750, # angle

    0.200, # inside

    0.200, # above

    0.200, # left-of


    0.125, # align1

    0.125, # align2

    0.125, # align3

    0.125, # align4

])
```



I was able to score 11/12 on the Basic Set C, thus showing the effectiveness of my agent when it came to 3x3 verbal progressive matrices. However, I suspect that throwing more visual heavy problems at it without verbal descriptions would be the next order of business in improving performance.

## PROBLEMRESULTS.CSV

Problem,Agent's Answer,Correct?,Correct Answer

Basic Problem B-01,2,Correct,2

Basic Problem B-02,5,Correct,5

Basic Problem B-03,1,Correct,1

Basic Problem B-04,3,Correct,3

Basic Problem B-05,4,Correct,4

Basic Problem B-06,5,Correct,5

Basic Problem B-07,6,Correct,6

Basic Problem B-08,6,Correct,6

Basic Problem B-09,5,Correct,5

Basic Problem B-10,3,Correct,3

Basic Problem B-11,1,Correct,1

Basic Problem B-12,1,Correct,1

Basic Problem C-01,3,Correct,3


Basic Problem C-02,4,Correct,4

Basic Problem C-03,4,Correct,4

Basic Problem C-04,8,Correct,8

Basic Problem C-05,3,Correct,3

Basic Problem C-06,7,Correct,7



Basic Problem C-07,2,Correct,2

Basic Problem C-08,1,Incorrect,5

Basic Problem C-09,2,Correct,2

Basic Problem C-10,7,Correct,7

Basic Problem C-11,4,Correct,4

Basic Problem C-12,3,Incorrect,8

Challenge Problem B-01,-1,Skipped,6

Challenge Problem B-02,-1,Skipped,1

Challenge Problem B-03,-1,Skipped,3

Challenge Problem B-04,-1,Skipped,4

Challenge Problem B-05,-1,Skipped,6

Challenge Problem B-06,-1,Skipped,3

Challenge Problem B-07,-1,Skipped,6

Challenge Problem B-08,-1,Skipped,4

Challenge Problem B-09,-1,Skipped,4

Challenge Problem B-10,-1,Skipped,4

Challenge Problem B-11,5,Correct,5

Challenge Problem B-12,1,Correct,1

Challenge Problem C-01,-1,Skipped,7


Challenge Problem C-02,-1,Skipped,7

Challenge Problem C-03,-1,Skipped,3

Challenge Problem C-04,-1,Skipped,8

Challenge Problem C-05,-1,Skipped,4

Challenge Problem C-06,-1,Skipped,7



Challenge Problem C-07,-1,Skipped,3

Challenge Problem C-08,-1,Skipped,1

Challenge Problem C-09,-1,Skipped,7

Challenge Problem C-10,-1,Skipped,3

Challenge Problem C-11,-1,Skipped,4

Challenge Problem C-12,-1,Skipped,2