Project 2 Reflection

"Solving 3x3 matrices as collections of 2x2 matrices"

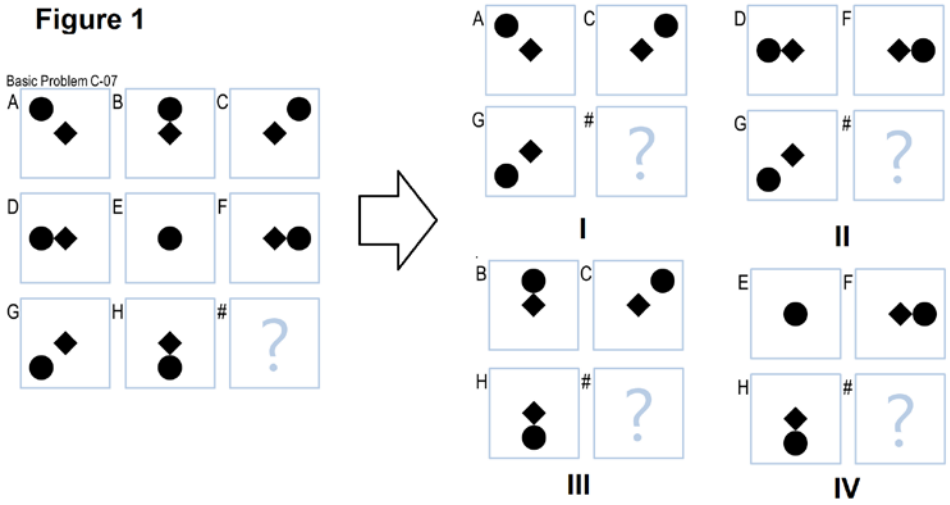
How does your agent reason over the problems it receives? What is its overall problem-solving process? Did you take any risks in the design of your agent, and did those risks pay off?

The design for solving 3x3 raven's progressive matrices is built on my previous agent which had successfully solved 12/12 of Basic Problem B and 11/12 Test Problem B. My previous agent took four steps to solve a 2x2 RPM problem:

1. Store each figures' objects in a list.
2. Pair up two lists of objects (list A and list B), and store their operation scores.
3. Pair up list C and list1 to 6, and store their operation scores separately.
4. Compare the differences of the operation scores and identify answer.

One 3x3 matrix can be seen as a collection of a series of 2x2 matrices as shown in Figure 1. I(A, C, G #), II(D, F, G, #), III(B, C, H, #), IV(E, F, H, #) and et al.

Figure 1



One major difference between my current agent and the previous agent is the comparisons and answers lists:

For 2x2 matrices:

comparisons = [["A", "B", "C"], ["A", "C", "B"], ["B", "C", "A"]]
answers = ["1", "2", "3", "4", "5", "6"]

For 3x3 matrices:

comparisons = [["A", "C", "G"], ["A", "G", "C"], ["C", "G", "A"], ["D", "F", "G"], ["D", "G", "F"], ["B", "C", "H"], ["B", "H", "C"], ["E", "F", "H"], ["E", "H", "F"]]
answers = ["1", "2", "3", "4", "5", "6", "7", "8"]

This seems like a simple idea, but the arrangements of comparisons deserve some thinking. I put *["A", "C", "G"]* and *["A", "G", "C"]* as the first and second comparisons in the comparisons list, thus if the 3x3 matrices are symmetrical, they can be solved at very beginning. And after each comparison, answers that do not fit will be removed from the answers list, which leaves less and less comparisons at late stages. The advantage of

this strategy is that it can greatly reduce comparisons during the whole run, and my agent usually does not need to finish all comparisons in the comparisons list before it delivers correct answer. The risk of this strategy is that if the correct answer is removed at earlier stages by mistake, then the search will fail after the agent finishes all comparisons in the comparisons list (there is no make up for earlier mistakes). That's another reason why the arrangement of comparisons in comparisons list is important (I do not want to exclude correct answer at earlier stages).

How does your agent actually select an answer to a given problem? What metrics, if any, does it use to evaluate potential answers? Does it select only the exact correct answer, or does it rate answers on a more continuous scale?

The strategy used to find correct answer is similar to what used in my previous agent. I created two help functions. One function is *operation* which compares two objects and returns a score, e.g unchanged: 5, rotation: 4, delete: 1. The other function *pairUp* which identifies paired objects in different object lists and returns a score list.

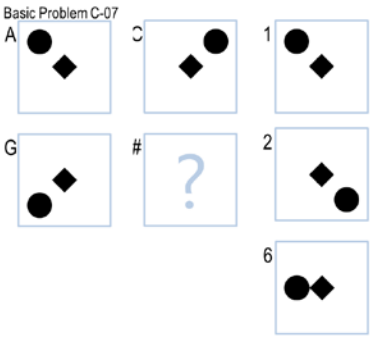
For example: by comparing A to C, I got a score list likes *[0, 5, 5]*, meaning one object changes shape and two objects keep unchanged. And by comparing G to 1-8, I got a list of score lists likes *[[0, 5, 5], [0, 0, 0], [1, 3, 4], [0, 0, 0], [0, 5, 5], [0, 4, 5], [1, 2, 4], [0, 5, 5]]*. By computing the difference between these score lists, the answers list will be updated as *["1", "5", "8"]*, and other answers that do not match are removed from the original answers list. The new answers list will undergo next round of comparison.

My strategy does not directly select correct answer, instead it removes incorrect answers from answers list until only one matched answer left.

What mistakes does your agent make? Why does it make these mistakes? Could these mistakes be resolved within your agent's current approach, or are they fundamental problems with the way your agent approaches these problems?

In the first a few runs, my agent only solved half problems in Basic Problem C. My agent inherits a big defect from the previous agent which ignores object location changes. For 2x2 matrices in Basic Problem B, due to the limited objects in each figure, the agent still found correct answers. But for 3x3 matrices in Basic Problem C, this defect is intolerant. As shown in Figure 2, answer 1, 2 and 6 are identical if we don't consider their position changes.

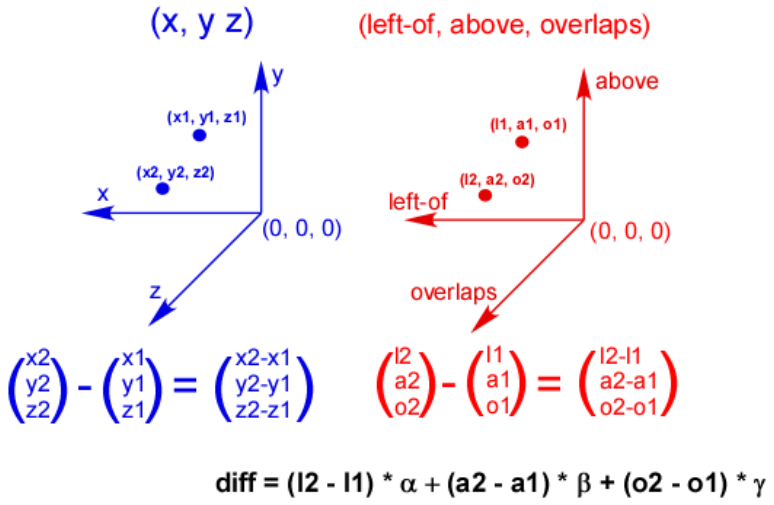
Figure 2



KBAI Fall 2015
Project2 Reflection

The strategy that I used to solve position changes is mimic to (but more flexible) how we calculate the difference between two points in a three dimensional matrix (Figure 3).

Figure 3



In a three dimensional matrix (x, y, z), there are two points: (x1, y1, z1) and (x2, y2, z2). Their difference can be represented as (x2-x1, y2-y1, z2-z1). Similarly, I construct a three dimensional matrix (left-of, above, overlaps). If one object is left-of four other objects, and above two other objects, it can be represented as (4, 2, 0). These information can be easily extracted:

```
loc = [0, 0, 0]
if "left-of" in obj.attributes:
    loc[0] = len(obj.attributes["left-of"].split(","))
if "above" in obj.attributes:
    loc[1] = len(obj.attributes["above"].split(","))
if "overlaps" in obj.attributes:
    loc[2] = len(obj.attributes["overlaps"].split(","))
```

For two objects: (l1, a1, o1) and (l2, a2, o2), their locations' difference can be presented as (l2-l1, a2-a1, o2-o1). And I further simplified it to one number using three parameters: α, β, γ :

$$\text{diff} = (l2-l1)*\alpha + (a2-a1)*\beta + (o2-o1)*\gamma.$$

The three parameters: α, β, γ can be refined, and I used $\alpha = 0.01$, $\beta = 0.03$, $\gamma = 0.05$ in my agent. The advantage of use small values to represent position changes is that I can first ignore them at earlier comparisons (by setting eps < 0.1 as identical) and only consider them at later stages when I still have multiple answers (by setting eps < 0.0001 as identical).

KBAI Fall 2015
Project2 Reflection

What improvements could you make to your agent given unlimited time and resources? How would you implement those improvements? Would those improvements improve your agent's accuracy, efficiency, generality, or something else?

My strategy is to remove answers that do not fit from the answers list after each round of comparison, thus my agent only need perform less and less comparisons. My agent usually does not have to finish all comparisons in the comparisons list before giving correct answer. The work flow is like:

For 3x3 matrices:
comparisons = [["A", "C", "G"], ["A", "G", "C"], ["C", "G", "A"], ["D", "F", "G"], ["D", "G", "F"], ["B", "C", "H"], ["B", "H", "C"], ["E", "F", "H"], ["E", "H", "F"]]
answers = ["1", "2", "3", "4", "5", "6", "7", "8"]
1: *["A", "C", "G"]* and *answers* (9 comparisons) → update *answers = ["1", "5", "8"]*
2: *["A", "G", "C"]* and *answers* (4 comparisons)→ update *answers = ["5", "8"]*
3: *["C", "G", "A"]* and *answers* (3 comparisons)→ update *answers = ["8"]*
4: return the correct answer *"8"* (total 16 comparisons).

My agent is efficient by using this strategy. It finished all problems in Basic Problem C in less than 5 seconds. Caveats: 1, if the correct answer is removed by mistake at any earlier stages, nothing can compensate that mistake. 2, the "correct" answer may be returned too early. If there is only one answer in the answer list, my agent will stop future comparisons and return the answer. What if a better answer is generated at later comparisons?

If I have unlimited time and resources, I would use a different strategy, and create a dictionary using answers as keys and their match times as the values. Finally, my agent picks the answer with highest score. The work flow is like:

For 3x3 matrices:
comparisons = [["A", "C", "G"], ["A", "G", "C"], ["C", "G", "A"], ["D", "F", "G"], ["D", "G", "F"], ["B", "C", "H"], ["B", "H", "C"], ["E", "F", "H"], ["E", "H", "F"]]
answers = ["1", "2", "3", "4", "5", "6", "7", "8"]
dict = { "1" : 0, "2" : 0, "3" : 0, "4" : 0, "5" : 0, "6" : 0, "7" : 0, "8" : 0}
1: *["A", "C", "G"]* and *answers* (9 comparisons),
update *dict = { "1" : 2, "2" : 0, "3" : 0, "4" : 0, "5" : 2, "6" : 0, "7" : 0, "8" : 2}*
2: *["A", "G", "C"]* and *answers* (9 comparisons),
update *dict = { "1" : 2, "2" : 0, "3" : 0, "4" : 0, "5" : 3, "6" : 0, "7" : 0, "8" : 3}*
……..
9: *["E", "H", "F"]* and *answers* (9 comparisons),
update *dict = { "1" : 3, "2" : 0, "3" : 2, "4" : 0, "5" : 4, "6" : 3, "7" : 0, "8" : 5}*
10: return the correct answer *"8"* (total 81 comparisons)

I can also sign different values for different comparisons, for important comparisons, I can give them higher scores. For example the matched answers from the comparison between *["A", "C", "G"]* and *answers,* their values will increase 2 instead of 1. Through further refinement, my agent will be much more accurate compared with my previous design but less efficient (81 comparisons vs 16 comparisons).

KBAI Fall 2015
Project2 Reflection

How well does your agent perform across multiple metrics? Accuracy is important, but what about efficiency? What about generality? Are there other metrics or scenarios under which you think your agent's performance would improve or suffer?

My agent solved 12/12 2x2 problems in Basic Problem B and 11/12 3x3 problems in Basic Problem C in less than 5 seconds. Since most properties of 3x3 matrices (either symmetrical, reflected or gradually changes) can be captured in the collections of 2x2 matrices, I am quite confident for my agent's generality. My agent's performance can be further improved by expanding the collection list of 2x2 matrices. For example, the 3x3 matrix in Figure 1, I can also include 2x2 matrices like V(D, E, H, #), VI(E, F, #, G), VII(F, D, H, #), VIII(D, E, #, G) ….

Which reasoning method did you choose? Are you relying on verbal representations or visual? If you're using visual input, is your agent processing it into verbal representations for subsequent reasoning, or is it reasoning over the images themselves?

I used verbal representation in project 2. For project 3, I plan to write some help functions to first transform images into verbal representations, then solve it using agent designed in this project.

Finally, what does the design and performance of your agent tell us about human cognition? Does your agent solve these problems like a human does? How is it similar, and how is it different? Has your agent's performance given you any insights into the way people solve these problems?

Similarity:

During the designing, I always ask myself that how we, humans, solve this kind of problem, and whether there are something that I can learn from. Two strategies that I learned from human cognition were implemented in my agent.

The first one was implemented in the help function *pairUp*. To compare objects from two figures, Human eyes always find identical objects first, and further identify left objects having transformations. In my agent, to pair up objects from two lists, instead of direct calculation of all permutations, I implemented similar strategy to identify unchanged objects first, then the objects with transformations.

The other was implemented to calculate object position changes. Instead of use exact x, y values to identify object position changes, I used a more flexible matrix that use the right bottom object as a starting zero (details illustrated in Figure 3). Same as human cognition. Let's assume that you show your friend a family photo and he asks who your brother is. You may say the left 2 instead of giving x, y values of the photo.

Difference:

There is a difference between my agent and human cognition. Humans more like to use Generate & Test strategy to generate the right answer, and compare the right answer with candidates, that's because human eyes can easily find identical objects, the generate & test strategy makes comparison super easy. But for computers, I found implementation of the generation step is difficult, while direct comparisons are quite simple.