CS 7637

Project 1 Reflection

Daniel Rozen

drozen3

GTid: 903104846

Project 1 Reflection

(questions being answered are in italics)

- *How does your agent reason over the problems it receives? What is its overall problem-solving process?*

  My agent reasons over the problems it receives by first representing all of the objects in the matrix and answers verbally in a Semantic Network. It then uses the Generate and Test method in order to loop through each potential answer (figures 1 to 6), placing the answer in figure D, and then measures the relationships and transformations from the objects in figures A to B and figures C to D and from figures A to C and figures B to D. It then compares these transformations to each other, comparing (A to B) to (C to D) and also (A to C) to (B to D).

  Since we often have multiple potentially close answers the agent then uses similarity weights in order to choose the best match. These were chosen based on importance of similarity.

- *Did you take any risks in the design of your agent, and did those risks pay off?*

  The risks I took in designing my agent was starting from the easier problems, which were the basic problems B-01 and B-02, and developing my Python code to solve them. Once my agent was able to find the correct matches for them I then went on to solve slightly

more difficult problems. This was in contrary to perhaps a better method of making a high level design that incorporates all of the problems and then brining it down to the code.

This risk paid off in that I was able to successfully code an agent that solves 10 to 11 out of 12 basic problems.

- *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?*

My agent uses metrics to assign a score to each of the potential answers, rating them on a continuous scale and then picks the best answer, which is the answer with the highest score. It does this by rating each answer, and then once done rating the answer, it checks if this is the highest rated answer yet, and selects it as the new best answer.

Transformations were recorded and stored in a dictionary named transformDict.

Different correct transformations are assigned different points in order to better be able to choose the best answer. I assigned different points for each of same general property

The correct transformation values were chosen based on the importance of similar transformations as I perceived them.

Here are the similarity weights that I chose:

```
sameIncr = .1  # same general property increment value
angleIncr = 10  # same angle rotation transform increment value
reflectIncr = 20  # same reflection transform increment value
```

fillIncr = 35 *# increment for correct fill transformation*

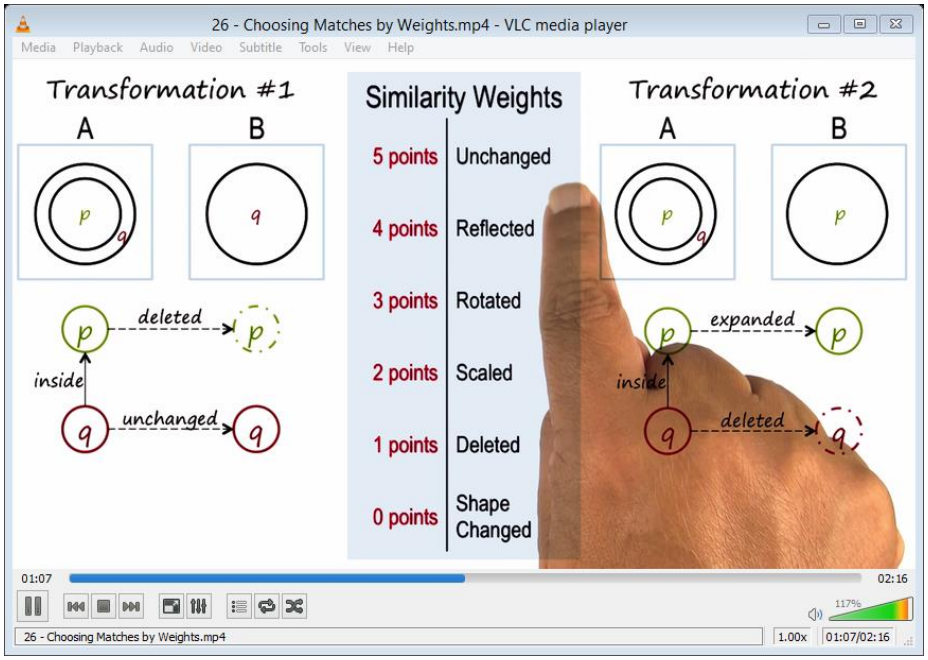
shapeIncr = 100 *# increment for correct shape transformation*

deleteIncr = 170 *# increment for correct deletion of objects*

alignIncr = 30 *# increment for correct alignment transformation*

These were chosen in the following order of importance: deletion, shape transformation, fill, alignment, reflection, rotation, and same properties.

This is somewhat similar to the Similarity Weights diagram in Lecture 3-26.



In order to solve basic problems 1 and 2, in which the correct answer is the same shapes as A, B, and C, my agent compared all possible answers with generate and test method and compared all objects with their respective attributes in answers 1 to 6 to the objects in A, B, and C. Every time there was an attribute match, that answer got a score incremented by *sameIncr* which was set to 0.1 points. The highest total score was chosen as the correct answer.

This method failed however for Problem 3 since there was an angle transformation from A to B and C to D. Therefore, I had to devise a method to compare transformations from A to B and C to D and from A to C and B to D.

To solve for this, I created a method in which I created a variable to measure the differences in angles from (A to B) and (C to D) and also (A to C) and (B to D). I then compared these differences comparing (A to B) to (C to D) and also (A to C) to (B to D).

If these differences matched, the total score for that particular answer was incremented by *reflectIncr*, which was set to 20 points.

This method however only worked for Problem 3. But it didn't work for subsequent problems. I then began to work on solving problem 4 and 7 which were both reflection problems with 1 shape. If first tried to solve by measuring the angle rotations and comparing them. However, I found that the correct answer for problem 4 didn't have the same transformations.

This is what my agent found:

abAngleDiff= -90

cdAngleDiff= 90

acAngleDiff= -270

bdAngleDiff= 90

I noticed that in these transformations, the differences are off by 180 degrees. So I made another comparison to see if abs(abDiff + 180) % 360 = abs(cdDiff) and abs(acDiff + 180) % 360 = abs(bdDiff). This solved for checking for correct reflections.

This worked for problem 4, but not for 7 due to the lack of choosing correct fill transformations. This was added next and worked for correctly choosing 7. If the shape fill changed across the transformation, the corresponding key in the dictionary was assign a 1. Otherwise it was assigned a 0. This correctly worked for problem 7.

However problems 6 and 9 were still failing due to the lack of priority for shape fill transformations. This was solved by increasing the weight of a correct fill transformation to 12, when compared to angle increase of 3 and reflection increase of 4 and other attributes of 1.

Next to be added was shape transformations, eg. from hexagon to heart or deleting the shape as was causing failure in solving problems 5, 8 and 10, 11, and 12. This was achieved by checking if an object changed from A to B etc., and if it did, concatenating a string of the object in each frame. Eg. P-08 A to B would be 'circlesquare'. This would be compared to C to D and if the same transformation was made, the score was incremented by shapeIncr = 100.

This worked for problems 8 and sometimes for problem 10.

2 was running into trouble because the objects are fed in in the opposite order so circleplus seems to transform to pluscircle. Therefore I had to add a check to make sure the same words weren't in the transform in a different order. I did this by comparing the 2 strings after they were **sorted**.

However 5 was remaining as object alignment transformations had to be applied. This was achieved by checking for which alignment transformation occurred in each dimension. eg. from bottom-right to bottom-left was a right-left transformation. This was done by concatenating a string of the alignment and checking for the differences and deleting the common words from the string. eg. "bottom-right-bottom-left" became "right-left" so that I could focus on the positional change only and make the appropriate comparisons across the figures. This solved problem 5.

Problems 11 and 12 had to take into account deletion of an object. This could be achieved by checking for the number of objects in each figure and if there's less in one figure, making a string of all of the shapes in each figure and checking for which shapes are missing. However, first I made an attempt to calculate the difference in the number of objects from A to B, C to D and then A to C and B to D. If the transformation was the same, I multiplied the point increase by the number of objects deleted. This avoided assigning scores to 0 deletions and also helped discern problems such as problem 12 where there were multiple deletions.

- *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?*

  My agent makes some mistakes. One of them is that it gives different answers for different runs. This is due to the fact that python dictionaries are not sorted, and the objects are fed in as a python dictionary in different orders. Therefore different objects are being compared in each run.

  This could be resolved within my agent's current approach by sorting the dictionaries using Python's class OrderedDict which is part of the standard python setup. Or use another method to ensure that the objects are always entered in the same order.

  Also, objects aren't sufficiently matched to each other when transformed because of the random order.

  At the moment the agent sometimes gives the wrong answer for problem 6 and always gives a wrong answer for 11 due to a bug in shape transformations, it always gives 6 instead of 1, which both have a deletion and similar shapes. 6 sometimes gives the wrong result of 4 likely due to a bug in fill transformations.

  What improvements could you make to your agent given unlimited time and resources? How would you implement those improvements?

Given unlimited time and resources, I would also add “size”, “above”, and “inside” transformation calculations in addition to what I already have. I would implement them in a similar way to my shape transformation.

I would ensure the same objects are passed in and make an improved comparison and matching of the objects. I would do implement this by checking for similarity among objects and similar transformations among objects and assigning correspondence values.

This would also likely fix the bugs in shape and fill transformations which I believe are due to incorrect object matching.

- *Would those improvements improve your agent’s accuracy, efficiency, generality, or something else?*

These improvements would improve my agent’s accuracy as there would be greater differences in the scores for answers as there would be more metrics to measure the best answer.

It would also improve generality as more metrics would more accurately determine the correct answer over more varied problems than the ones given.

Efficiency would slightly go down as there would be more checks for transformations with every loop. Also each object passed in would have to be checked for the correct corresponding object. However, this would be barely noticeable for our purposes as the agent runs in about 1 second.

- *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 does reasonably well across multiple metrics, but sometimes gives the wrong answer based on matching up the wrong objects which is based on the order the objects are fed in. This can hurt generality.

Efficiency is pretty fast for our purposes, taking less than a second to compute. However, there are many double nested loops which may slow down the computation over large data sets.

Other scenarios such as ones with many shapes may cause my agent's performance to suffer as there would be a higher chance of object mismatches.

- *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 relied on verbal reasoning method as explained above.

- *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?

The design and performance of my agent tell us the following about human cognition: That our minds make many computations, decisions, and interpretations without us being aware. Designing the agent forced me to break the transformations down to simple steps instead of just solving the figures by merely looking at the problem and deciding which answer looks correct.

My agent somewhat solves the problem like a human does.

It is similar in that it makes a step by step comparison between the figures and its neighbor to see what transformations are made in order to find the best solution. Eg. It first checks shape change, then fill, then rotation, etc. However, it is more similar to people in more complicated problems where a step by step comparison is required. In simple problems such as B-01, a quick scan of the whole picture gives an easy answer for a human.

It is different as it randomly loops through all objects in the figure and compares them, whereas a human will naturally see much easier which objects map to which. Also a human is able to take the whole picture into account, such as in problems 3 and 4, whereas the agent only looks at limited transformations from one figure to its neighbor.

My agent's performance has given me some insights into the way people solve these problems. One is that people easily see how the answer fits into the big picture. Humans

can more easily correspond objects based on quickly seeing shape, fill, angle, position across figures. Also humans sometimes make multiple transformations at the same time, eg. shape and fill at the same time.