Rodrigo De Luna Lara

Knowledge-based AI: Cognitive Systems

Project 1 Reflection

02/06/2017

1. **What is your agent's reasoning approach - visual or verbal? Does your agent need to convert between one or the other?**

The designed agent uses a mixed approach, for every problem, a verbal representation is constructed using a class named Frame, which has 2 properties, the name of the frame (A,B,C,1,2, etc.) and all of its objects under the *object* property. To illustrate the construction of the verbal representation of each problem, let's take a look at a frame from a problem (Figure 1).

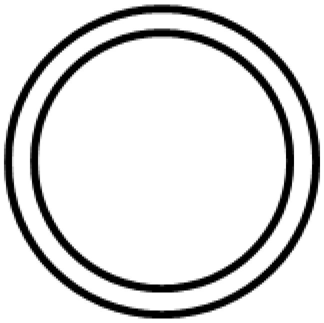

Figure 1. Frame A from Basic Problem B-10.

This frame has the following content in the problem data:

```
A
        a
                shape:circle
                size:very large
                fill:no
                inside:b
        b
                shape:circle
                size:huge
                fill:no
```

On the first level there is the name of the frame (‘A’), on the second level there are the labels for each object in the frame (‘a’ and ‘b’), and on the third level the properties for each object in the form *key:value*.

For each problem, the agent iterates over each frame and creates a feature matrix by iterating over each object, the attributes which are passed to the matrix contained in the *object* property of the *Frame* class. Due to the fact that not every object across the different problems has the same attributes in their figures, a dictionary of attributes was created to ensure proper representation of the attributes of each frame:

name	shape	size	fill	angle	inside

Note that not all objects have all of these properties, if an object doesn’t have any of these properties, they are set to *None*. For the frame in Figure 1, the resulting verbal representation is:

```
[[‘a’ ‘circle’ ‘very large’ ‘no’ None ‘b’]
[‘b’ ‘circle’ ‘huge’ ‘no’ None None]]
```

Therefore, each frame can be associated directly to the concept of the same name seen in class, being composed of several objects individually linked to instances of objects with common attributes, as seen in Figure 2. In such a way, each object follows a template, enforcing consistency.

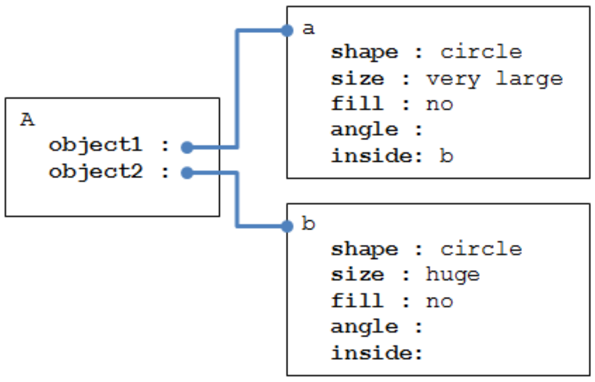

Figure 2. Composition of each frame by object sub-frames.

As for the visual representation, the frames are loaded as Pillow images. This allows for direct processing of the frames as Pillow objects. All processing of the visual representations is done using the Image, ImageOps and ImageMath modules of Pillow. For each problem, both the visual and verbal representations are built, and depending on the transformations detected, they are used in different ways to obtain a solution to the Raven's Problem, no conversion is necessary between them.

2. How does your agent represent/store the images efficiently? What is your agent's overall problem-solving process?

As explained in the question above, the representation for each image is stored as a feature matrix. Having the representations as a matrix allows for easier comparison between objects, as the appropriate rows or columns can be processed by their index to determine transformations. The semantic network can be obtained by performing logical operations between matrix elements; for example, for Basic Problem B-01 (Figure 3) the resulting representation for frames A,B and C is:

```
[['a' 'square' 'very large' 'yes' None None]]
[['a' 'square' 'very large' 'yes' None None]]
[['a' 'square' 'very large' 'yes' None None]]
```

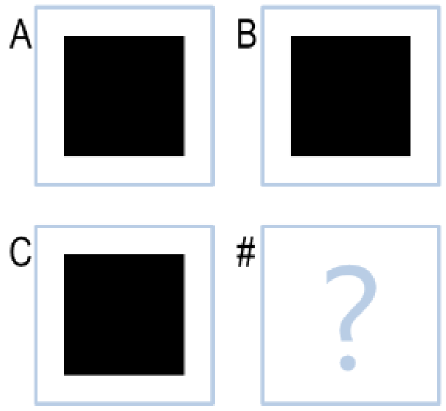

Figure 3. Problem B-01.

Using the verbal representations as matrices allows for simple logical operations between them. The simplest case is when nothing changes, like in Figure 3, so if the objects of both frames are the same then there is no transformation in any of the attributes of the frame

```
if (frameA.objects[i] == frameB.objects[i]):
   transformAB[i] = ‘unchanged’
```

Therefore, the knowledge representation the agent creates, as a matrix, is very efficient and allows calculating the transformations with less effort.

As for the visual representation, the current version of the agent doesn’t identify the objects in each frame; it treats the image as a whole. This may seem limiting, but it allows for quick transformations such as mirroring or flipping. The frames are processed as whole images so the transformations are simple and effective when used.

3. Provide an overview of the design of your agent

The agent uses basic semantic networks to determine how to solve the problem. As explained before the objects of each frame have the same properties and they can be easily compared to determine the following transformations:

Condition	**Transformation**
if shape changes	‘shape-change’
if fill changes	‘fill-change’
if angle changes	‘rotated’
if object count changes	‘added’/’deleted’
if nothing changes	‘unchanged’

Coupling the resulting transformation with the knowledge representations of frames A, B and C results in a very crude semantic network. This semantic network is used to generate the solution for the missing frame, and then this prototype solution is compared against all possible solutions. Depending on the transformation, either the agent looks up for a complete match on the object description (for verbal processing) or for the frame which results in the highest accuracy in comparison with the prototype solution (for visual processing).

The current design of the agent has the following flow chart:

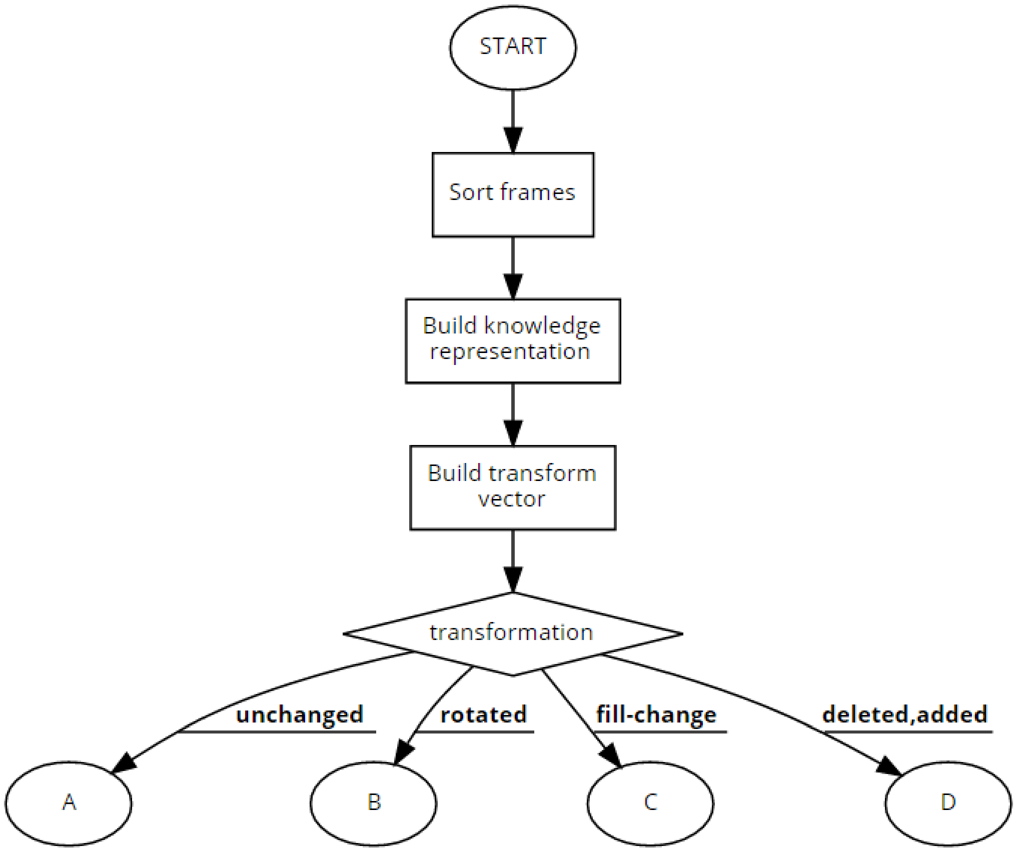

Figure 4. Flow diagram for agent's start.

The agent starts by sorting the frames (letters first), creating the semantic network by first creating the feature matrix from the objects' attributes and then determining the transformations between frames. Depending on the predominant transformation, the agent uses one of 4 methods to determine the solution.

The first case is when the transformation matrix is 'unchanged', i.e. when the problem is the most trivial problem (like Problem B-02 in Figure 5), when ⅔ or all frames are the same. In this case the feature matrices are compared to match the solution, as shown in Figure 6.

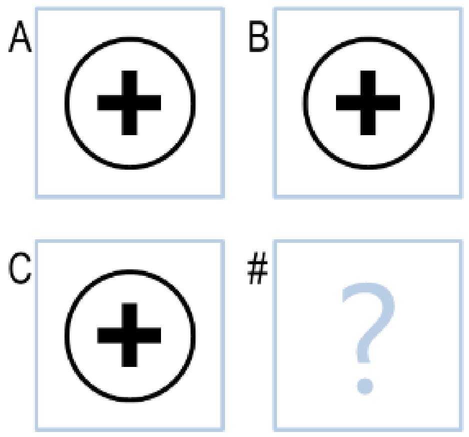

Figure 5. Problem B-02.

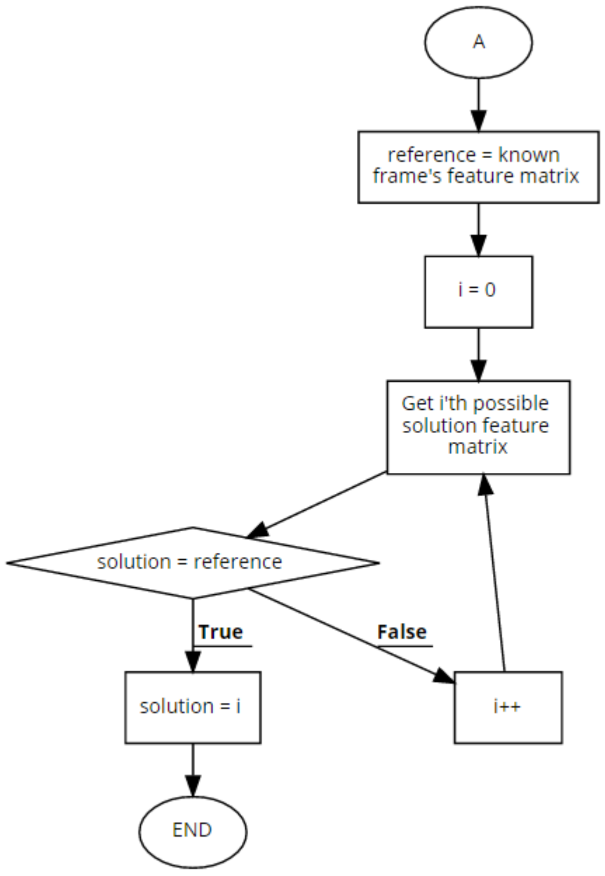
A
reference = known frame's feature matrix
i = 0
Get i'th possible solution feature matrix
solution = reference
True
False
solution = i
i++
END

Figure 6. Unchanged processing flow chart.

The second possibility the agent considers is if the image was rotated, considering so far only mirroring and flipping of the images (as seen in Problem B-04, Figure 7) as the only possible transformations. In this case the agent uses the visual representation to determine the solution, as seen in Figure 8.

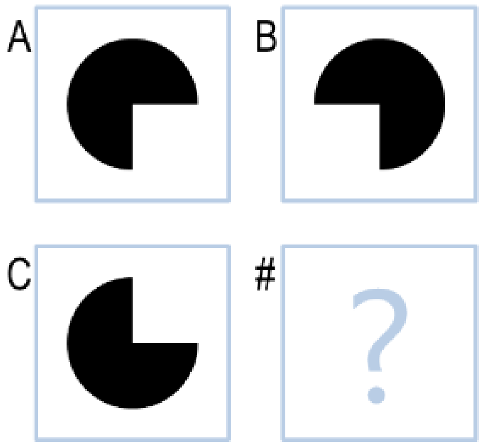

Figure 7. Problem B-04

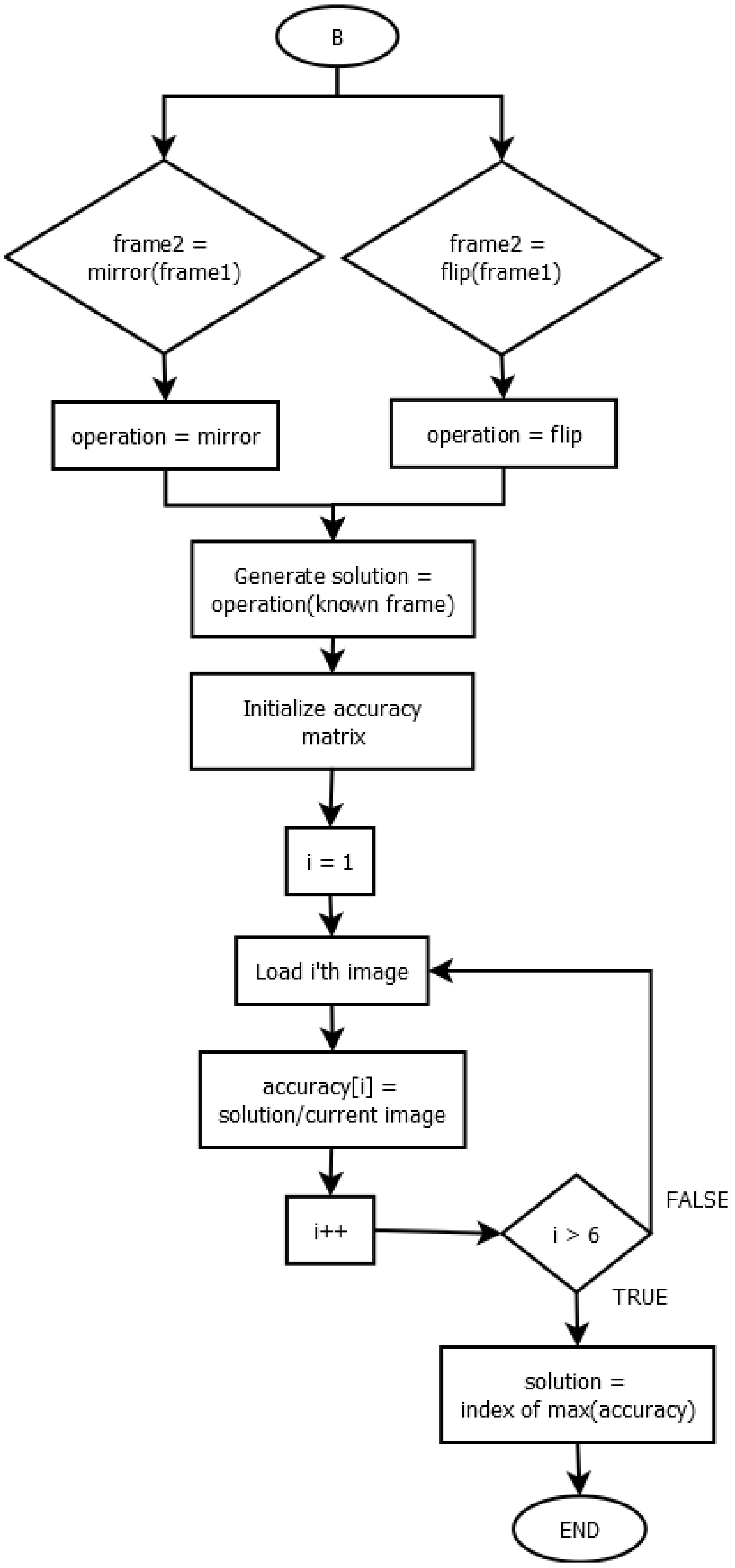
B
frame2 = mirror(frame1)
frame2 = flip(frame1)
operation = mirror
operation = flip
Generate solution = operation(known frame)
Initialize accuracy matrix
i = 1
Load i'th image
accuracy[i] = solution/current image
i++
i > 6
FALSE
TRUE
solution = index of max(accuracy)
END

Figure 8. Rotated processing flow chart

The agent operates on frame A to determine if either frame B or C is a mirror or flip of frame A, then with this information it generates the appropriate solution. It then compares each of the available solutions with the generated solution and chooses the one that matches with the highest accuracy.

The third possibility the agent considers is if the fill of the figures change. This method was based on problem B-07 (Figure 9), and follows the procedure outlined in the flowchart in Figure 10.

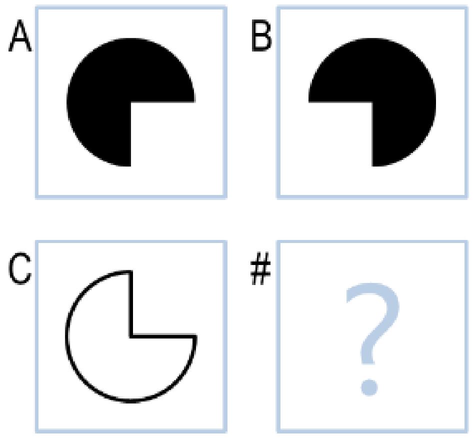

Figure 9. Problem B-07

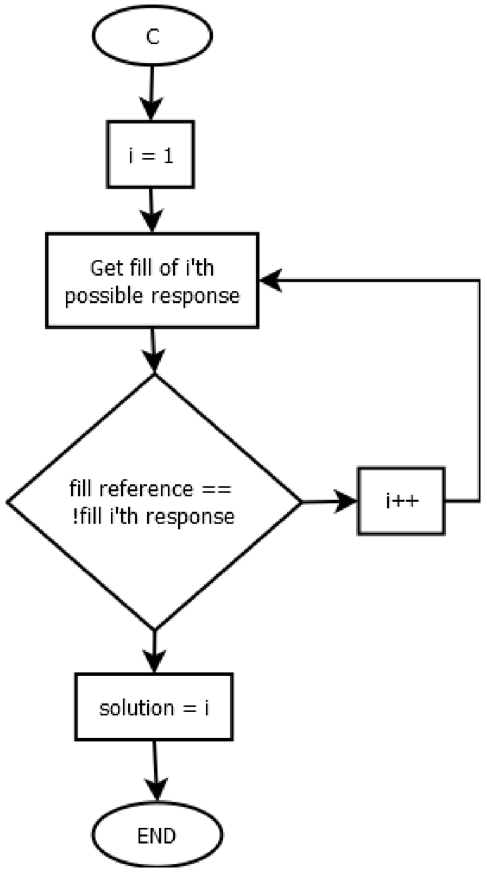
C
i = 1
Get fill of i'th possible response
fill reference == !fill i'th response
i++
solution = i
END

Figure 10. Fill change processing flow chart

The agent looks for the image with the opposite fill, returning the numbered response which correctly emulates the original transformation as per the semantic network.

The last case the agent considers is when an element is added or removed from a frame; such is the case of Problem B-11 (Figure 10). The agent uses the ImageMath module in these cases, as shown in the flowchart in Figure 12.

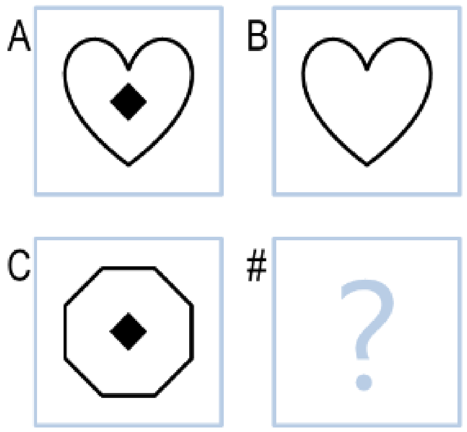

Figure 10. Problem B-11

It is possible to see what the agent does with Problem B-11, if the agent loads Image A and B and subtracts them using the Image Math module. This operation can be visualized in Figure 11.

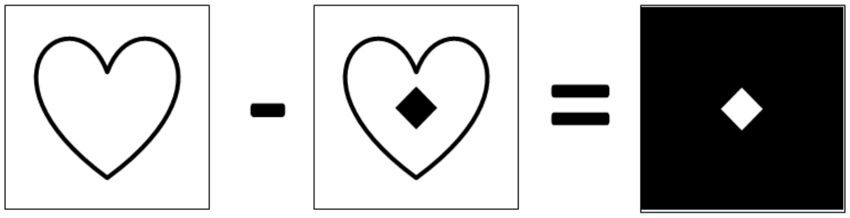

Figure 11. Subtraction of Frames B and A in Problem B-11

The resulting frame can then be used to generate the solution using Frame C, as seen in Figure 12:

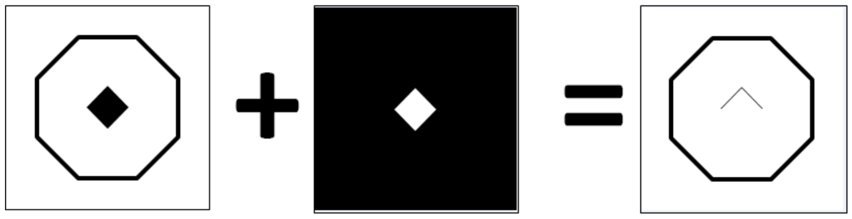

Figure 11. Generation of the solution for Problem B-11

It can be seen that the generation of the solution is not exact, due to misalignment between frames or jitter in the images; however, the generated solution is really close to the image of the actual solution.

This method emulates what a human would do. A human would look at the images and immediately determine which shape is being added or removed, without needing the knowledge representation of the objects. The agent determines what is being added or removed from the images, and then simulates the possible solution, finally picking the solution that best matches this simulation.

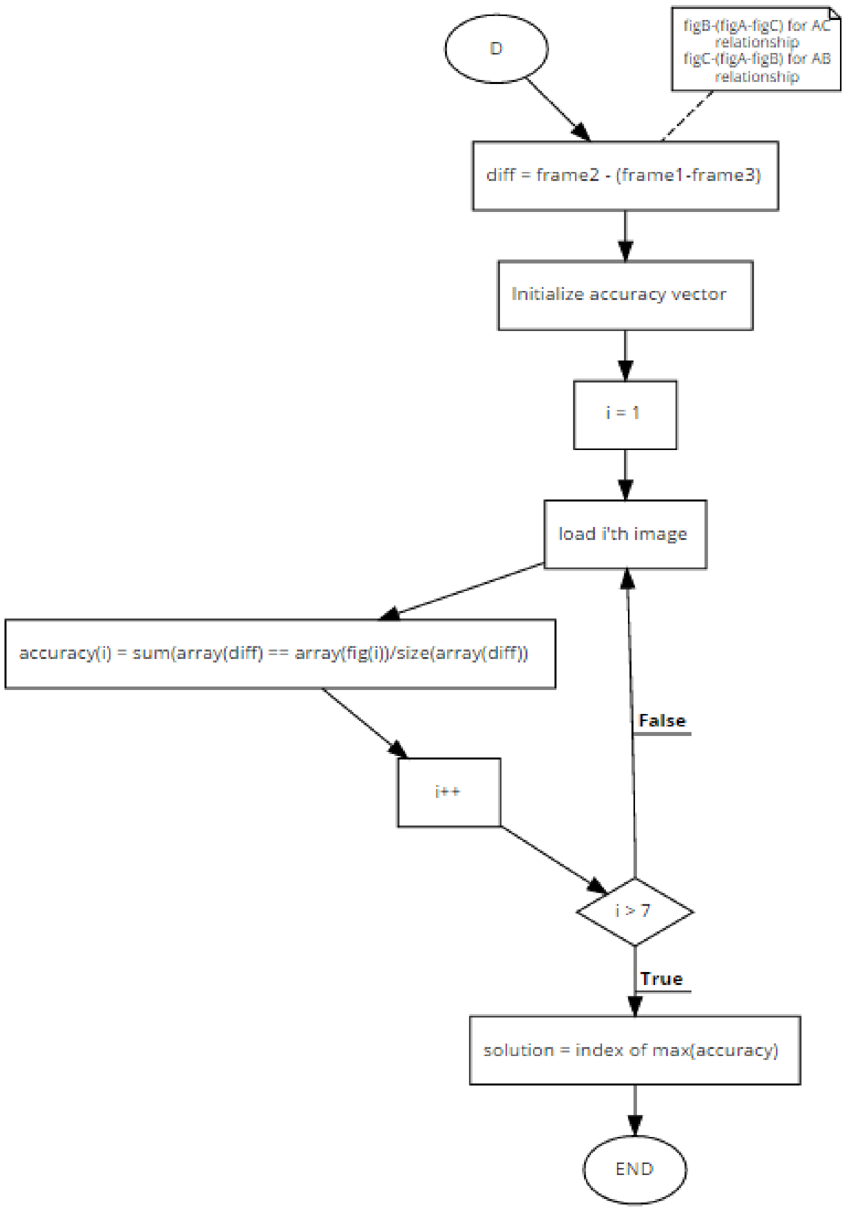
D
figB-(figA-figC) for AC relationship
figC-(figA-figB) for AB relationship
diff = frame2 - (frame1-frame3)
Initialize accuracy vector
i = 1
load i'th image
accuracy(i) = sum(array(diff) == array(fig(i))/size(array(diff))
i++
False
i > 7
True
solution = index of max(accuracy)
END

Figure 12. Added/deleted processing flow chart

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

The agent makes a lot of assumptions that could end up in mistakes, and some of the methods are not very robust. For example, the fill-change transformation only accounts for complete change in fill in the shape. If only a part of the image is having its fill changed, the current method won't be able to solve the problem adequately.

Another mistake the agent could make is choosing a suboptimal solution in the case that more than 1 of the methods it is programmed to perform are used in a same problem. In this case the agent uses the last solution produced by any of these methods, and is unable to recognize this situation entirely.

As the agent was programmed on the existing cases in Basic Problems B, it may not consider all possibilities of similar problems, so if the problems it receives are significantly different than the ones used to program it, there is the possibility that the agent may be unable to come up with a solution.

To resolve these mistakes, the agent's methods must be tweaked and expanded, as the fundamental issue is that they are very dependent on the available problems. Additionally, the current use of the semantic networks is very rough, perhaps investing more time on generating a generalized solution methodology which just looks at the verbal representations, determines the transformations and then determines which of the possible solutions best approaches the original setting would make the agent more general.

5. Please detail on your evaluation/performance criterias and your agent's results. Think about accuracy, efficiency and generality. Are there other metrics or scenarios under which you think your agent's performance would improve or suffer?

The performance criteria for the agent in this project was simply how many problems it was able to solve correctly, divided in training and testing sets, as is common in machine learning applications. The final results were that the agent has a 100% accuracy on the training set (12/12 problems correct) and 83% accuracy on the testing set (10/12 problems correct). This reflects the issue presented in the last question, the agent has some issues with previously unseen cases, as it is too specific in some of the methods. The accuracy was deemed acceptable for the current implementation.

Regarding the efficiency, the computation time was not recorded, and it was deemed unimportant as the agent is able to solve all 12 problems from the training set in a few seconds. On the other hand, talking about generality, the agent was deemed to be too specific, given the significantly lower accuracy on the testing set, so it has moderate generality.
The performance of the agent is justified by its design. As was mentioned before, the agent has a very simple implementation of semantic networks, and instead uses a more case-based reasoning approach. The agent was specifically trained with aims at solving correctly the problems in the testing set, this results in some cases being missed as they were not present in the training of the agent, hence the higher accuracy in the training set. The agent would greatly suffer if the problems are very different from the ones in the training set for this same reason.

6. Please provide an explanation of how your methods/components/ideas in your agent's design are/might be similar to (or can be related) to specific KBAI methods discussed in class.

Some of the previous questions cover the relationship between the agent’s design and specific KBAI methods. First, the images composing the Raven’s Progressive Matrix were represented as frames, having a specific set of attributes to define them uniformly across problems. The verbal representation of each problem is parsed to a specific arrangement in a feature matrix to ensure consistency. It was only natural to use Frames as knowledge representation in the agent, as it is a very natural way for programmers to define objects and classes in general. This is contradictory with human cognition, as it isn’t so structured and strictly defined as a Frame is, but for programming Frames make things easier to process and understand.

Being the frames the knowledge representation of the problem, they were used to produce a transformation vector to complete a basic semantic network. There are 2 semantic networks created by the agent before attempting to solve the problem, one for the relationship between frames A and B, and another for frames A and C. Using semantic networks at first was due to the fact that the transformations can be easily used to determine the solution (in some cases, like the trivial cases and fill-change cases). For some other problems they weren't deemed so effective so other methodologies were used.

Specifically, case based reasoning was used in general throughout the design of the agent by leveraging previous knowledge on problem solving using programming, as well as thinking how a human would solve the problem, later converting that to code. As in the case of Frames, case based reasoning is a natural skill for many programmers, programs in general tend to be trained on specific examples and then generalized, using previous knowledge to most efficiently solve them.

Finally, the methodology for Generate and Test was used in each method by the agent, the solution was generated and then the possible solutions were tested against it to determine which was the response to the RPM problem. In some cases it made processing very easy, like the visual processing for the last 3 basic problems, in which a prototype solution could be easily created and then tested against the available solutions to determine the response.

7. What does the design and performance of your agent tell us about human cognition? 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 agent emulates the cognition of its creator, therefore its performance is directly related to the ability of the programmer to first solve the problem, then make it abstract enough so that it can be represented programmatically and then solved. The issue here arises with the first step, what if the human struggles solving the Raven's Progressive Matrices from a start? The agent will certainly reflect this.

The agent emulates the reasoning process a human follows to solve a RPM problem, in the following sequence: first, the problem is observed and its elements are identified, then it must be determined if the solution to the problem can be found either by rows or columns in the matrix. Once having identified the relationship, a prototype solution is built (in the mind of the human) and then it is compared with the available solutions to see which one matches.

If the agent simulates the problem solving of the human programming it then it is subject to the same biases in the cognitive process. If the agent's creator fails to think of a possible variation in the problems, then the agent won't know about this special case. The agent in this project is programmed mostly by case-based reasoning, and has little ability to predict for unknown cases which deviate considerably from the cases used to create it.

This is a problem humans face too when faced with new types of problems. For example, in any mathematics class, the learning is done mostly by solving a large number of cases, learning from the mistakes to improve the ability to solve that particular set of problems. There is always a theoretical explanation on how things work, but most people will only truly learn how to solve a specific kind of problem by iterating over several problems and learning from the mistakes.

The agent in this project was trained with only 12 previously unseen (by the human) problems, and when abstracted programmatically there also exists a complexity factor related to the ability of the human in the specific programming languages used (Python or Java). Therefore, there is a skill bias in the agent related to its programming, adding an extra layer of complexity. As humans we have perform a myriad of complex operations effortlessly in our minds, AI agents have a more limited ability and skill set than humans, and so it can be concluded that while indeed the agent follows the same cognitive process as humans, it is limited by both its own limitations in the complexity of its methods, which in the end is dependent on the skill of the human to program it, so the performance of the agent is heavily biased because of this.

Sources

1. Goel, A.K. Semantic Networks (class material). Georgia Tech, College of Computing. February 2017.
2. Goel, A.K. Generate and Test (class material). Georgia Tech, College of Computing. February 2017.
3. Goel, A.K. Frames (class material). Georgia Tech, College of Computing. February 2017.
4. Goel, A.K. Case-Based Reasoning (class material). Georgia Tech, College of Computing. February 2017.
5. Lundh, F and Clark A. Pillow (PIL Fork) Handbook. Retrieved from https://pillow.readthedocs.io/en/4.0.x/handbook/index.html