Jonathan Satria
January 31, 2015
Project 1 Report - KBAI
jonathan.satria@gmail.com

Project 1 Report

1.0 Introduction

In Project 1 we are tasked with designing an AI agent to solve 2x1 Raven's Progressive Matrix problems. The implemented approach solves 18 of the 20 basic problems and is able to solve Challenge Problem 02. The project proved to be an excellent learning experience by applying the topics of semantic networks, correspondence problem, and problem solving methods in a practical setting. In this document I outline the approach taken to implement an AI agent that is able to solve these problems. I will also discuss the weaknesses, challenges, and possible improvements for the AI agent.

2.0 Implementation

Design and Approach

The AI agent I have designed follows three basic steps:

1. Generate a Semantic Network between A → B frames
    a. Correspond nodes A → B
    b. Generate the transformations from A → B
2. Apply transformations in Step 1 to frame C to generate the D, the "candidate" answer
    a. Correspond nodes in A and nodes in C
    b. Apply the transformations
3. For each answer choice, say X, compare and measure the similarity of D to X.

It is clear that of the topics covered in the class so far, I was able to utilize **semantic networks** to represent the relationships and transformations between A → B. I then used **means-ends analysis** (or perhaps a more condensed variation) to generate the transformations. After applying those transformations to C to generate the candidate answer choice, D, **generate and test** is used to "generate" each answer choice and "test" it against the candidate answer. Lastly, the fact that each step depends on the step prior is indicative of the **problem-reduction** method. Each step is now going to be discussed in further detail to reveal how the AI agent is able to solve the RPM problems.

1. **Generate a Semantic Network between A → B Frames**

Before discussing the individual substeps of generating the semantic network, I would like to define the semantic network within the context of the AI agent I have designed. In class we discussed a semantic network that contains two types of relationships - spatial

relationships and transformational relationships. It will become apparent the discussion that spatial relationships were not captured within the actual code, but instead remained an attribute of the object. At first I implemented a spatial relationship, but the entire semantic network appeared to be over-engineered. Since the spatial relationships are captured within the object attributes, I felt that it would be simple enough to refer back to the attributes if these relationships were ever needed.

Thus the semantic network that the AI agent builds in the next two substeps has:

- Nodes that correspond to objects within a frame
- Transformation relationship(s) that represent the changes of an object from A → B

a. Correspond nodes A → B

The first necessary step is to correspond the objects in frames A and B. This was a difficult task, as I decided not to rely on the object names and instead on the features and spatial relationships of the objects. The process of finding all possible arrangements of objects in A and B was very tedious and my initial approach was computationally inefficient. Essentially, I found all possible permutations of objects in A as a list (order matters) and arranged them to a fix set of B elements. Then I measured the similarity of the corresponding objects by using the similarity weight matrix. Below is pseudocode that outlines the basic steps of solving the correspondence problem.

```
// Method to correspond the nodes in A and B frames
public correspond (RavensFigure A, RavensFigure B){
      A_objects = A.objects;
      B_objects = B.objects;
      //get the arrangements possible for A and B objects
      List arrangements = getArrangements(A_objects, B_objects);
      for each arrangement{
            similarity = getSimilarity(arrangement);
            if similarity > currentSimilarity{
                  currentSimilarity = similarity;
                  currentArrangement = arrangement;
            }
      }
}

// Method to evaluate the similarity of a semantic network
public int getSimilarity(A_objects, B_objects){
      int similarity = 0;
      for each object in A_objects and B_objects{
            //apply similarity weight matrix
            if SHAPE_CHANGED
                  sum += 0;
```

```
            else if DELETION
                  sum += 1;
            else if SCALED
                  sum +=2;
            else if ROTATION
                  sum +=3;
            else if UNCHANGED
                  sum +=5;
      }
      return similarity;
}
```

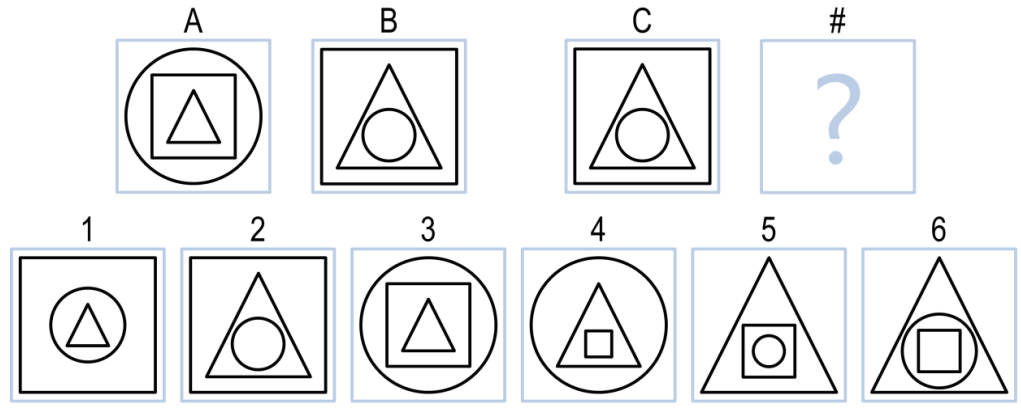
The method **correspond()** will iterate through each possible arrangement (as determined by **getArrangements**()) and evaluate whether that arrangement correctly corresponds nodes. That evaluation is provided by **getSimilarity()** which applies the similarity weight matrix to determine if nodes match. The weight matrix I used is depicted below:

Points	**Transformation**
5	Unchanged
~~4~~	~~Reflected~~
3	Rotated
2	Scaled
1	Deleted
0	Shape Changed

Please note that “Reflected” was not applied in this first project. I will discuss this caveat later in the Limitations section.

To demonstrate a significant amount of complexity, I will use the 2x1 Basic Problem #12 to illustrate this process.

2x1 Basic Problem 12



The input to the problem is as follows:

Frame A	**Frame B**
A Z shape:circle size:large Y shape:square size:medium inside:Z X shape:triangle size:small inside:Y,Z	B Z shape:square size:large Y shape:triangle size:medium inside:Z X shape:circle size:small inside:Y,Z

If we use the object names as *labels* of the objects in memory, then the possible permutations of A are ZYX, ZXY, XYZ, XZY, YZX, and YXZ. Beginning with the ZYX arrangement, getSimilarity will asses for each object the similarity weight. That method would yield the following results

Node in Frame A	**Node in Frame B**	**Similarity Weight**
Z shape:circle size:large	Z shape:square size:large	0 (shape changed)
Y shape:square size:medium inside : Z	Y shape:triangle size:medium inside:Z	0 (shape changed)
X shape:triangle size:small inside:Y,Z	X shape:circle size:small inside:Y,Z	0 (shape changed)

Total Similarity Weight: 0

If we repeat this process for the arrangement ZXY

Node in Frame A	Node in Frame B	Similarity Weight
Z shape:circle size:large	Z shape:square size:large	0 (shape changed)
X shape:triangle size:small inside:Y,Z	Y shape:triangle size:medium inside:Z	2 (scaled)
Y shape:square size:medium inside : Z	X shape:circle size:small inside:Y,Z	0 (shape changed)

Total Similarity Weight: 2

ZXY is better than ZYX, but if we continue our process for the rest of the arrangements we come upon the arrangement YXZ in which case the following similarity is assessed

Node in Frame A	Node in Frame B	Similarity Weight
Y shape:square size:medium inside : Z	Z shape:square size:large	2 (scaled)
X shape:triangle size:small inside:Y,Z	Y shape:triangle size:medium inside:Z	2 (scaled)
Z shape:circle size:large	X shape:circle size:small inside:Y,Z	2 (scaled)

Total Similarity Weight: 6

If this process is repeated for all arrangements, eventually it will be determined that the arrangement YXZ of A nodes to ZYX in B nodes is the correct pairing. We have now solved the subproblem of corresponding nodes.

b. Generate the transformations from A → B

The next step of the process of building the A → B Semantic network is to generate the transformations that occur. Since the AI agent has determined an “optimal” arrangement and correspondence of nodes in A to B, generating the transformations is a matter of recording the differences in the B, the goal state, and A, the current state. In my opinion, this

process is similar to *means-ends analysis*, but differs in that all of the transformations are discovered in one step instead of an iterative approach.

To determine the transformations, for each node in B I compared the attributes to the corresponding node in A. I then stored the transformations and associated them with the node in A. Below is the pseudocode of this process

```
public getTransformations(Node a, Node b){
      List transformations;
      if (FILL changed)
            transformations.put(TransType.FILL, b.attributes.FILL);
      else if (ANGLE changed)
            transformations.put(TransType.FILL, b.attributes.ANGLE);
      else if (SCALE changed)
            transformations.put(TransType.FILL, b.attributes.SIZE);
      else if (SHAPE changed)
            transformations.put(TransType.FILL, b.attributes.SHAPE);
      else if (a == null and b != null)
            transformations.put(TransType.ADDITION, b);
      else if (a != null and b == null)
            transformations.put(TransType.DELETION);
}
```

This method was repeated for each “pairing” of nodes in the A and B object lists. The product of this process was a HashMap of Nodes and Transformations. Specifically and technically speaking, this was represented as **`HashMap<Node, Transformations>`** where Transformations is a **`HashMap<String, String>`** of transformation types and values.

Referring back to 2x1 Basic Problem 12, the transformations would be depicted as follows:

A Frame Objects	**B Frame Objects**	**HashMap<Node, Transformations>**	
		Node	**Transformations**
Y shape:square size:medium inside : Z	Z shape:square size:large	Y	SIZE, LARGE
X shape:triangle size:small inside:Y,Z	Y shape:triangle size:medium inside:Z	X	SIZE, MEDIUM
Z	X	Z	SIZE, SMALL

shape:circle size:large	shape:circle size:small inside:Y,Z		

The AI agent has now generated the transformations of A that result in the objects in B

2. **Apply transformations in Step 1 to frame C to generate the D, the "candidate" answer**

The next major problem to be solved is applying the transformations to frame C. The AI agent must first correspond the nodes in A and C to determine where to apply the transformations. Then applying the transformations is a somewhat trivial task.

a. Correspond nodes in A and nodes in C

Initially the approach to corresponding nodes in A to C was similar to corresponding nodes in A to B. It soon became apparent that in some cases, like problem 12, the correspondence algorithm was not corresponding correctly due to pure reliance on object features. It was neglecting the spatial relationships of objects within a frame. So to correspond nodes in A and C, I measure the feature and spatial attributes of a particular arrangement of nodes. The highest similarity, of course, wins. Below is the pseudocode of this process. It is similar to the correspondence method previously, but note the additional step of evaluating the spatial similarity of nodes.

```
// Method to correspond the nodes in A and C frames
public correspondAC(RavensFigure A, RavensFigure C){
      A_objects = A.objects;
      C_objects = C.objects;
      //get the arrangements possible for A and C objects
      List arrangements = getArrangements(A_objects, C_objects);
      for each arrangement{
            //get the spatial similarity
            similarity = getCorrelationSimilarity(arrangement);
            if similarity > currentSimilarity{
                  currentSimilarity = similarity;
                  currentArrangement = arrangement;
            }

            //get the feature similarity
            similarity = getFeatureSimilarity(arrangement);
            if similarity > currentSimilarity{
                  currentSimilarity = similarity;
                  currentArrangement = arrangement;
            }
      }
}
```

```
// Method to evaluate the correlation similarity of a semantic network
public int getCorrelationSimilarity(A_objects, B_objects){
      int similarity = 0;
      for each object in A_objects and B_objects{
            //apply similarity weight matrix
            if spatial similarity equivalent
                  similarity += 5;
      }
      return similarity;
}
```

If we take ZYX as an arrangement of A to the ZYX of C, we get the following results

Node in Frame A	Node in Frame C	Spatial Similarity Weight	Feature Similarity Weight
Z shape:circle size:large	Z shape:square size:large	0 (no spatial attributes found)	0 (shape changed)
Y shape:square size:medium inside : Z	Y shape:triangle size:medium inside:Z	5 (inside: Z)	0 (shape changed)
X shape:triangle size:small inside:Y,Z	X shape:circle size:small inside:Y,Z	5 (inside: Z)	0 (shape changed)

Total Similarity Weight: 10

If we repeat this process for all permutations, we will find that ZYX has the highest similarity weight and is thus accepted as the answer. Note that weighing 5 points for nodes with similar spatial relationships was not a completely arbitrary assignment. The reasoning behind this is that if an object is unchanged, it gets 5 points. In a similar sense, if an object has the same spatial similarities, it too is *unchanged*.

b. Apply the transformations

Now that the AI agent knows which nodes to apply the transformations to, it simply takes each each A object and its transformations and applies them to the corresponding C node. The pseudocode is depicted below:

```
public void applyTransformations(Object c, Transformations){
      Object d = new Object(c);//start with c attributes
      for each transformation in Transformations{
            if (transformation type is SCALE)
                  d.attributes.SCALE = transformation.getValue();
            else if (transformation type is ROTATION)
                  d.attributes.ANGLE = transformation.getValue();
            else if (transformation type is FILL)
                  d.attributes.FILL = transformation.getValue();
            else if (transformation type is VERTICAL_FLIP)
                  d.attributes.VERTICAL_FLIP =
transformation.getValue();
            }
      }
```

The method generates a new object **d**, for each of the objects in C by applying the transformations. The end result is a collection of objects that should represent the answer. Again, using Ravens Problem 12 as an example, the AI agent would get the following results:

Node in Frame C	**Transformations**	**Nodes in D Candidate Answer**
Z shape:square size:large	SIZE, SMALL	Z shape:square size:small
Y shape:triangle size:medium inside:Z	SIZE, LARGE	Y shape:triangle size:large
X shape:circle size:small inside:Y,Z	SIZE, MEDIUM	X shape:circle size:medium

3. For each answer choice, say X, compare and measure the similarity of D to X.

In this last step of the process, the AI agent must compare the ideal candidate answer "D" to each answer X. The agent does this by a similar process as the correspondence problem, where nodes are corresponded and evaluated for their similarity. The variation here is that the similarity weight matrix favors equality of attributes only. In other words, no similarity is awarded if D and X have objects have rotation but at different angles.

Answer 1 in Problem 12 would result in the following (assuming correspondence has been resolved)

Node in Frame D	**Nodes in Answer 1**	**Similarity**
Z shape:square size:small	Z shape:square size:large	1 (Shape)
Y shape:triangle size:large	X shape:triangle size:small	1 (Shape)
X shape:circle size:medium	Y shape:circle size:medium	2 (Shape, size)

Total Similarity: 4

Now if the AI agent evaluates answer choice 6

Node in Frame D	**Nodes in Answer 1**	**Similarity**
Z shape:square size:small	X shape:square size:small	2 (Shape)

Y shape:triangle size:large	Z shape:triangle size:large	2 (Shape)
X shape:circle size:medium	Y shape:circle size:medium	2 (Shape, size)

Total Similarity Weight: 6

The AI agent, seeing that answer 6 has the highest similarity to the candidate answer then announces that 6 is the correct answer.

Generating Set of Transformations
Types of transformations -

Node to Node

- Rotation (Rotate or flip?)
- Scale
- Deletion
- Addition
- Fill (all or none, some parts)
- Shape Changes: Subtract edges (Challenge Problem 01)

Frame

- Shape Changes: Add nodes to get sum of edges

Applying

2.3 Limitations & Weaknesses

As mentioned in the introduction, the AI agent discussed thus far is able to correctly answer 18/20 Basic problems and 1 of the challenge problems. Though successful on many accounts, there are certain limitations and weaknesses that became apparent when trying to solve more challenging or ambiguous problems. Some of the limitations and weaknesses will be discussed briefly here, before addressing them in a larger discussion in section 3.

Rotation vs Reflection

Basic Problem 7, 13, and 18 became difficult to answer correctly altogether because 7 and 18 represent rotations whereas 13 represents a reflection. I was unable to implement a effective strategy to attempt both rotations and reflection when transforming the C nodes.

Ambiguity

Basic Problem 16 also posed challenges because the correct transformation is for the inner object to be expanded while the outer object deleted. This is in contrast to basic problem 4 where it is a simple deletion of the inner object. I was able to solve this temporarily by reattempting the correspondence step where scaling is favored over an unchanged object. Unfortunately, I was not satisfied with this approach because the implementation required twice the computational time to process the answers.

Frame Transformations vs Node Transformations

In Basic Problem 18, the correct answer is to apply the “rotational” transformation to all nodes of C, regardless of the fact that A and B contain less objects than C. This is seemingly obvious to a human, but was in fact very difficult to approach in my AI agent implementation. The agent here assumes that nodes transform *individually* and disregards that an entire frame can undergo a transformation. In fact, the aforementioned rotation versus reflection problem could have been approached by restricting reflections to entire *frames* (such as Problem 13).

Similarity Weight Measures

Admittedly, there is a bit of non-science and ambiguity in how the similarity values are attributed. I would have liked to been able to apply a more sound and mathematical approach. One problem that the AI agent has is that there are three different correspond() methods, each with different ways of going about determining similarity.

These are just some of the perceived weaknesses and areas of improvement for the AI agent, but if addressed the AI agent could have more effectively approached the RPM problems.

3.0 Discussion

3.1 Improvements

This approach was fairly straightforward and is not significantly different than the approach I had in mind prior to beginning the project. I am satisfied with the results and am admittedly surprised at how well the AI agent performed given such a straightforward approach. There are many areas of improvement though, that I hope to incorporate in the next iteration.

One-Pass Approach

The current AI agent implementation only goes through the problem solving method once with a constrained set of procedures, similarity weights, and correspondence approaches. One consequence of this is that the agent is unable to “try” different transformations. For example,

if the AI agent were able to try reflection instead of rotation, problem 13 would have solved. Additionally, in Challenge Problem 01, if the agent were able to retry the transformation “shape change” and instead apply the transformation “divide number of sides by 2”, it would have discovered that the triangle is the correct answer. The AI agent is rigid and seems to exhibit a lot of *deductive* reasoning rather than the qualities of an *intelligent* agent.

Improved Knowledge

The level of detail that was abstracted and captured in the generated semantic network was sufficient to answer 18/20 RPM problems, but given more language it would have been able to solve at least problem 13. In this specific context, had the agent known that the angle of a rotation of a triangle is 60 degrees, it would have known that a 180 degree rotation *may* result in the same image. In a more general sense, it would have been helpful to apply some knowledge about the symmetry and rotational properties of the various objects.

In addition, it would have also been helpful to include the spatial attributes in the semantic network. Move operations were not represented in the semantic network but will definitely be utilised in the next iteration of the AI agent.

3.2 Final Thoughts

This first project has demonstrated that a capable AI agent needs to be flexible in thinking about a problem. Topics of learning have already come up in class lecture and I am enthusiastic and curious to see how such practices can improve the AI agent. Given that KBAI is a new venture for me (a web developer), I can see how my approach is rigid and deductive in nature - like an HTTP request and response structure. I would like to see how I can implement various RPM problem solving strategies and incorporate them into one AI agent that is able to correctly choose the right answer. I wanted to refrain from coding conditional statements to process particular problems separately (although admittedly I did this for problem 18). I would like to implement the problem solving in a more generic and abstract sense such that solving a set of RPM problems does not require complicated branching.

In summary, Project 1 has been a fantastic learning experience and I am enthusiastic and eager to apply new topics to make the AI agent better.