**Project Report**

**Project 4**

*by*

Jassimran Kaur

jassimran@gatech.edu

Georgia Institute of Technology

CS 7637- Knowledge Based AI – Cognitive Systems

Report for Project 4

Apr 12, 2015

**Overview**

The mind's mechanism for storing and retrieving knowledge is fairly transparent to us. Computers, unfortunately, are not adept at forming internal representations of the world. Instead of gathering knowledge for themselves, computers must rely on human beings to place knowledge directly into their memories. This suggests programming!

In this paper-

* Firstly I will describe the reasoning used by my agent to answer Raven's progressive matrices and how does it work.
* Secondly, I will elaborate how my agent comes to some of its correct answers and why the agent makes some of the mistakes it does.
* Thirdly, if I had more time and resources, what I could have done to improve it.
* Next, I will describe the unique challenges and opportunities in visual reasoning compared to verbal reasoning
* Lastly, I will describe the relationship between my final agent and human cognition, with focus on the visual problem solving: does your agent solve problems the way humans do?

**Architecture of the agent**

Before I dig into details of how my agent works, I want to briefly describe the overall architecture of my agent and reasons for using this approach. Architecturally, it is divided into different components like Corresponder, Generator, Tester, Comparer etc. Each of these components can be reused. I am reusing some of these components for 3x3 problems as well. OpenCV is used to create RavensProblem data structure from Project 2. This data structure is then fed to the agent for usual processing.

2. Agent sends RavensProblem to establish correspondence

Agent

Corresponder

Comparer

Generator

Tester

3. Comparer stores transformations

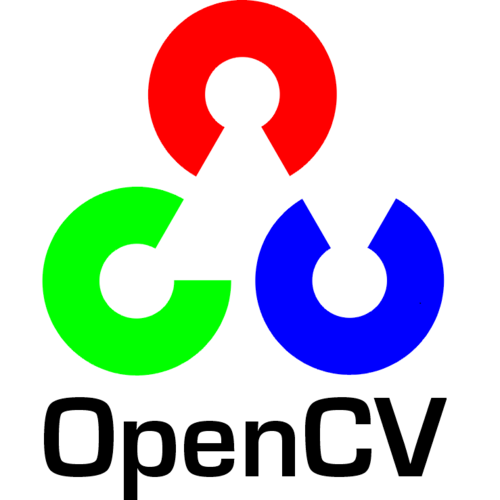
4. Generator applies transformations and generates best possible solution

5. Tests each choice with generated solution

1. Agent reads images using OpenCV

Output: Best match

Weight each option



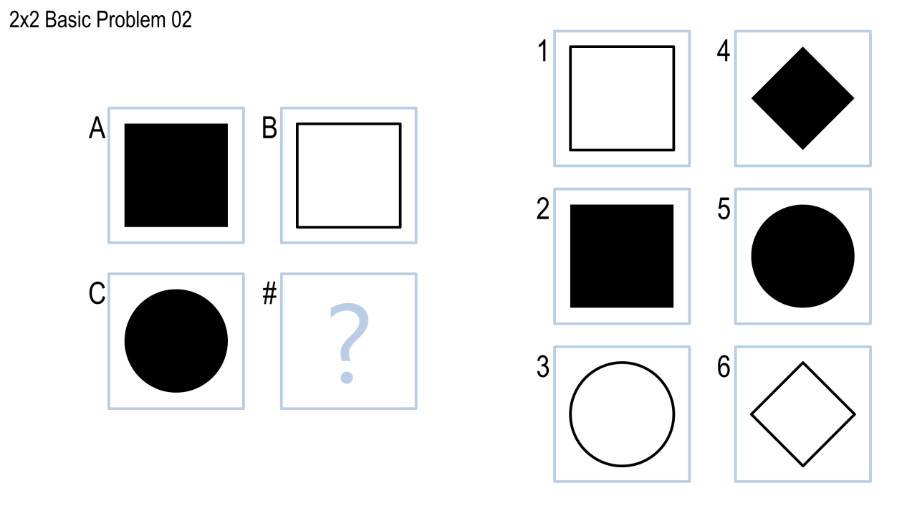
RavensProblem

**Image 1: Conceptual View of the****Agent**

**Working of the agent**

Before starting the programming for my agent, I reflected upon how I was solving the problem. Let’s see the approach my brain took to solve the problems:

1. I started with problem as shown below in figure 2:



**Image 2: Basic Problem 2x2 - 2**

1. I observed how Image A transforms to Image B, its shape remain same but it gets unfilled. I was visually able to establish the correspondence among the images by seeing the properties like shape, fill, size etc.
2. Then I observed how Image A transforms to Image C, its shape changes but fill remains same.
3. Then I considered the problem image - Image A and combined the two transformations and applied to A i.e. the shape should change and it must be unfilled to get expected answer.
4. I compared my answer with each of the option for similarity.
5. The one which resembled my solution most was chosen the correct answer.

Agent can’t perceive but if provided with data and methods, they can process the data to come to conclusions. My agent uses **OpenCV** for recognizing the images, **Semantic** **Networks** for knowledge representation and **Generate & Test** method to manipulate the data.

I thought of leveraging the amazing features and variety of algorithms available in OpenCV to recognize the images. The reason behind choosing Semantic Networks is its simplicity, its ability to make the relationships explicit, transparency, and speed as it doesn’t save extraneous data and high computability. Generate & Test method first guesses the solution and then tests whether this solution is correct and satisfies the constraints. Due to limited resources, my agent uses Smart Generator which is well-informed and generates the best possible solution. The tester then tests the solution against every option. This increases performance of the Agent. This method is known to work very well for the simple problems and seemed very promising.

Let’s see the usage of these methods in our Agent by following the steps below:

1. Agent gets the images using *imread* method in OpenCV and for each Figure, *recognizeImage* method returns *RavensFigure* as shown below.

Mat image = Highgui.imread(path, Highgui.CV\_LOAD\_IMAGE\_GRAYSCALE);

RavensFigure figure= recognizeImage(image, image\_name);

**Pseudo code 1: Reading images using OpenCV**

1. *“recognizeImage”* method does all the processing by recognizing the contours of each of the shapes inside the Figure. It recognizes shapes like square, triangle, circle, pac-man, arrow and etc and each of their properties like size, fill, angle. It also finds the position like above, left-of, inside and sets all the properties for the object. Intricacies of this be discussed later in “Getting correct answers section”. Agent accumulates this data and creates a RavensProblem which is passed on to the Corresponder component, which helps in establishing correspondence among images.
2. It calls “getMatch” method and compares each object in A with every object in B. The maximum matching object is chosen as the corresponding object and marked as “Matched”. The next object in A finds its best match and so on. There are some special cases in which sufficient number of objects do not exist in A or B or C. They are handled by comparing the number of “matched” objects counts with original count. While establishing correspondence maximum weightage is given to similarity of shape.
3. Corresponder returns a Hash Map data structure as shown below:

HashMap <String, HashMap <Integer, HashMap<String, String>>> finalMap

**Pseudo code 2: Establishing correspondence within Images**

1. First, let’s see the next steps for 2x1 and 2x2 problems. Agent calls the comparer which takes corresponded data in “finalMap” and compares all attributes in A to corresponding attributes for corresponding object in Image B and C. For example, attribute “shape” in object 1 of image A will be compared to attribute “shape” in object 1 of image B if its exists.

For each object in A { //like 1, 2, 3

For each object in B {

If (name\_of\_object\_A equals name\_of\_object\_B) { //like 1 in A & B

For each attribute in object\_A { //like shape, size, fill

For each attribute in object\_B {

If (attribute\_name\_A equals attribute\_name\_B) {

Compare(attribute\_value\_A with attribute\_value\_B)

Store transformations in a Map data structure

}

}

}

}

}

}

**Pseudo code 3: Establishing correspondence within Images**

1. The comparison between the different attributes of objects help establish the relationships. For example: Change of angle from 0 to 180 degree is termed as *rotation180clockwise* transformation.

Fill\_Comparer (attribute\_value\_A with attribute\_value\_B) {

If attribute\_value\_A = attribute\_value\_A: return “same”

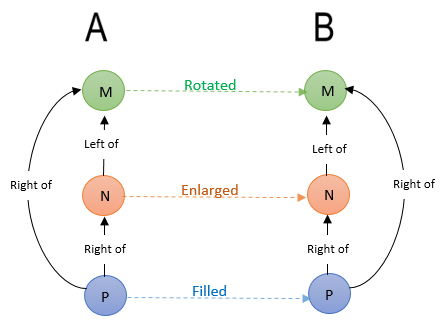
Else if attribute\_value\_A = “no” and attribute\_value\_A = “yes”: return “filled”

Else if attribute\_value\_A = “yes” and attribute\_value\_A = “no”: return “unfilled”

}

**Pseudo code 3: Comparing A with C**

1. All these transformations help form the sematic network from A->B and A->C and is saved in a Map data structure for 2x2 problems and from A->B for 2x1 problems.



**Image 2: Formation of a semantic network**

1. The Agent then goes ahead and applies the transformations from A->B on C for 2x1 problems and A->B *XOR* A->C for 2x2 problems and generates a best possible solution D. This is step 1 of Generate & test method. Due to limitation of resources, Agent utilizes a smart generator to get D. Pseudo code for generating a sample D is shown below.

For each object in A {

If there exists a transformation in A->B & A-> C, Map for that object\_A {

For each attribute\_A in object\_A { //like shape, size, fill

Apply the transformation stored in A->B

XOR

Apply the transformation stored in A->C

Generate value for attribute\_D

*//Example: Transformation for size=enlarged && attribute\_A\_Value=small Generate attribute\_D in object\_D with value=large*

}

}

}

Add the generated values for each attribute to expected Map\_D.

**Pseudo code 4: Generation of correct answer D**

1. For 3x3 problems, generating a solution was little tricky and different because there are so many transformations like A->B, A->C, D->E, D->F, G->H, B->E, C->G and etc. totaling 28 in number. I was able to identify some patterns in these problems, hence I decided to leverage those patterns to come to answers. Few of the identified patterns are:
2. *All objects are same*
3. *All objects are same horizontally*
4. *All objects are same vertically*
5. *All objects are same diagonally in South-East direction*
6. *All objects are same diagonally in South-West direction*
7. *Number of objects increase arithmetically horizontally*
8. *Number of objects increase arithmetically vertically*
9. *All objects increase arithmetically in size*
10. *All objects rotate by x degree horizontally*
11. *All objects rotate by x degree vertically*
12. *If all shapes are same, Sum of angle of A & B figure is angle of C & Sum of angle of D & E figure is angle of F,*
13. *And many more……..*
14. Extensively using if-else clause, code has been written to recognize all these patterns and make decisions based on these selected pattern. For example, for pattern 11 mentioned in above text box, generated solution is Sum of angle of G & H.
15. In step 2 of Generate & Test method, we test our above generated solution with each of the given answer choices 1-6. Based on attribute type certain score is added to the total score for the option.

Loop option 1-6 {

For each attribute\_option in each object\_option { //like shape, size in Z,Y,X

If Map\_D contains attribute\_option && attribute\_option\_value = attribute\_D\_Value

Then Add +2 to total\_score of option

Else If Map\_D does not contains attribute\_option

Then Subtract -1 from total\_score of option

}

Add total\_score for each option to score\_Array

}

**Pseudo code 4: Tester for D and each option 1-6**

1. Once the score of each option 1-6 is known, the option with maximum score is the right choice and hence the answer. For example, my agent gives following score for Problem 3 mentioned in Figure 1. Hence Option 4 with score [6] is the correct choice.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Option | Option 1 | Option 2 | Option 3 | Option 4 | Option 5 | Option 6 |
| Total  Score | [4] | [4] | [2] | [6] | [2] | [4] |

**Table 1: Score for options 1-6**

**Correct answers and mistakes**

To make my agent *learn* the relationships and transformations, I started the way I *learn* things – Start with simplest problem first and then grow from there to complex problems.

Since, I am leveraging my code from Project 1 & 2, I will try to focus on explaining how my agent recognize the shapes using OpenCV first and then I will move forward on explaining the mechanism once the images have been read correctly.

I used OpenCV method – “*findContours*”, which finds the contours. For each contour, “*approxPolyDP*” method returns the number of edges in each contour and these edges helps decide the shape of each object. I use “*isContourConvex*” to check if the image is convex or not in combination with no of edges to decide the shape.

|  |  |  |
| --- | --- | --- |
| No of edges |  | Shape |
| 3 | Yes | Triangle |
| 4 | Yes | Square/Rectangle |
| 5 | Not | Half-Arrow |
| 7 | Not | Arrow |
| 12 | Not | Plus |
| 8 or above | Yes | Circle |

Once shapes were determined, the properties of each object like fill, size, angle etc were determined. Since there is no standard method available for any of these properties, these values from derived based on the geometric features of each object. Hierarchy data structure filled by “*findContours” method* was used to determine the fill in combination with other parameters. For example, area of the contour was used to determine the size of the shape. The positional attributes like inside, above, left-of and etc. were determined by utilizing the position of centroid of each object.

The second step was to feed this data to the agent from project 2, which determines the solution. This functionality is explained in previous report for Project 2, so I will not go into its details.

To get desirable output a lot of tweaks were used. For example, in some shapes double use of “*approxPolyDP*” method with varying accuracies is used to establish the correct number of edges. Centroid of figures was always approximated because even same looking shapes had different values of centroids. Since such details are difficult to be perceive my eyes, humans do not need to approximate. Angles of square were determined by using “*minAreaRect”* method and it was helpful in most of the cases.

Yet, this process was not enough to get all the problems right. The agent makes some mistakes due to number of reasons, listed below:

* The problems containing triangles and having changes in orientation of triangle do not work as required. It is mainly due to the fact that my method which finds the orientation of triangle isn’t reliable and it works only half the time. It’s not impossible to make it more accurate but due to time limitations, larger epsilon value of “*approxPolyDP*” due to less resources and varying sizes & types of triangles hindered in achieving it. 2x1 Basic Problem 15, 2x2 Basic problem 16, 2x2 Basic Problem 20, 3x3 Basic Problem 17 all fail due to this problem.
* My agent fails to identify the images which do not have clear cut edges, for example a circle inside a square is easy for a human to perceive but OpenCV library method recognizes it as four concave blocks. Such problems are still a challenge.
* Some problems which have partially filled sections are contributing to mistakes because my shape recognition algorithm ignores the outer contour, if it has child contour. Hence some shapes get ignored adding to inconsistencies in answers. This scenario is occurring in 2x2 Basic Problem 06.
* A lot of problems fail because of problem of correspondence which was introduced due to shortcomings of Project 1 and Project 2. For example, 2X1 Basic Problem 16 is simple, but due to problems in establishing correct correspondence between triangle & inner circle, it gives an error.
* There were some challenges in addressing the problems in which angles change. Mathematically, a symmetrical object if rotated by 180 degrees doesn’t changes shape. For Example: A square when rotated by 90 degrees remains the same. A circle doesn’t have any effect of rotation on it. Thus such rules were added to the Agent to take care of specific cases of rotation. A plus if rotated by 45 degrees or 315 degrees will still look the same. Also concept of diagonal symmetry was introduced when some images flipped by 180 degrees diagonally.
* Similarly, fill could be partial fill or full fill. Partially filled object can have top-half filled or left-half filled or even top-left or bottom-left filled. This has been established using the centroid of the un-filled object. This further helps in recording the correct transformation. But this is not working for a partially filled triangle like 2x1 Basic Problem 10
* My agent is not yet ready or programmed to tackle problems in which a transformation has been applied to one object but then in another transformation, that object changes position.
* It is not able to solve 2x2 basic problem 17 as it doesn’t recognize that overlapping squares if all filled will result in a one big filled square.
* For 3x3 problems, once the pattern has been recognized, the appropriate solution is generated. If a new pattern that has not been covered in the patterns comes into picture, it will give the incorrect answer.

**With more time and more resources**

Working on this project was challenging but a fun experience. However one can get easily carried away by intricacies of a particular problem and spend hours and hours on it, without much output. That is exactly what happened in my case. There is so much I wanted to do but couldn’t do because of time constraints and limited resources. I am glad we have this section.

1. Learning OpenCV was challenging and I did learn a lot. But OpenCV is huge and I never got time to fully explore all its features and hence it’s full potential. I would like to explore its features and utilize them in finding the properties of objects and making the data more accurate. For example, I want to improve the accuracy of values of angles in almost all shapes. And I also want to find a better way to distinguish between filled contours and unfilled contours.
2. First weakness of the agent is its dependency on only one method to solve the all of the problems i.e. Generate & Test. I would like to use another method to the problems namely - “Means-Ends Analysis”. ‘Image A’ is ‘Start State’ and ‘Image B + Image C’ is the ‘Goal State’. All the transformation to reach “goal state” must be saved and applied on C to generate a solution D. This solution should then be tested against all six options to get the best match. The matching score for each option must be saved to an array.

*Note however, that I intent to use this method in conjunction with already implemented “Generate & Test” method.*

|  |  |  |  |
| --- | --- | --- | --- |
| Option Name | Score from Generate & test method | Score from Mean-Ends Analysis method | Total Score |
| *Option 1* | *-2* | *2* | *0* |
| *Option 2* | *0* | *2* | *2* |
| *Option 3* | *3* | *6* | *9* |
| *Option 4* | *2* | *2* | *4* |
| *Option 5* | *0* | *4* | *4* |
| *Option 6* | *3* | *3* | *6* |

To arrive at the final answer, we will use the total score of both the methods as shown in table 3 below. Based on this table, option 3 gives the best result. Note however, Generate & test alone would have not been sufficient in this case. This adds to confidence level of the agent.

**Table 3: Total Score**

1. I wanted to implement learning in my agent and make it smart enough to learn from its mistakes and store the successful cases. One approach I thought of is to use “checkAnswer” method and store each rule. But due to lack of resources that is very tough to implement. I then planned on using only the failed solutions to implement learning. But due to lack of time, effort could not be put into this task.
2. I would also like to use Frames in combination with semantic networks as a data structure for processing the transformations. However, changing the core structure was a major task. As of now, my code uses a complex HashMap data structure which is not very efficient and rather cumbersome to use. In future I will use a custom frame class to save the values of attributes for each object.
3. I didn’t get the chance to refactor the code and make it more presentable.
4. My Corresponder component isn’t very accurate. I would like to use improve it. Rather than comparing first object with all the objects in a figure, I would rather build a more Robust Corresponder. Most of the failing problems in current project are due to failure of Corresponder.
5. There are way too many patterns in 3x3 problems, I would like to invest so time in generalizing them and eliminating duplicates.

**Unique challenges and opportunities in Visual Reasoning**

While doing this project I have started appreciating human brain more and more. We can distinguish between images in blink of an eye but making a machine recognize the same needs a lot of programming. It was an exciting experience but it came with its own set of challenges.

Firstly, it was difficult to decide what sort of approach should be taken to tackle this project. Since I had a pretty good running agent from project 2, I decided to go for recognizing the images using a library.

One of the challenges in visual reasoning is that a shape, let’s say a triangle can be represented in thousand forms but visually it still looks like a triangle. For example, it can be an obtuse triangle, right-angled triangle, equilateral triangle, inverted triangle etc. To provide an abstraction for these factors, I thought of converting the visual data to textual data. This provided for limited textual attributes to deal with. It also saved processing time and hence became an opportunity for improving the project.

Another challenge in visual reasoning was to distinguish the importance of an attribute in solving the problem. For example, in a problem where size of none of the objects in figures change doesn’t need to have size property. It is only an overhead in comparing the objects. It was very difficult to decide such properties. I chose to go with four primary properties – shape, size, angle and fill and making remaining properties optional and added only when they are significant in deciding the correct answer.

A big problem was that the images which look exactly the same to humans aren’t actually equal precisely. Thus computationally comparing the images needed a lot of approximations. A square with 15000 units of area is large and another square with 15500 area is also large. Human eye will easily categorize them as large but a machine will need to be explicitly told that any area with +- 500 units are same.

A tightly packed circle inside might look like four concave blocks connected at edges. Identifying such images accurately is a challenge.

**Relationship between my final agent and human cognition**

Human cognition and agent are similar in a lot of ways. Just the way humans recognize patterns among changing angles, changing shapes, changing positions and etc, so does my agent. It has the set of rules which helps it decide that if particular pattern is seen then follow corresponding solution. For example: When a square turns by 90 degree, humans see that the shape remained same and angle is reset to zero. Similarly agent can be programmed to recognize such patterns. If a human doesn’t know a rule, he will also fail in implementing that rule in agent.

Once taught to tackle a specific situation, agent will always answer correctly, however humans are still prone to error. Another situation arises when a situation is slightly different, humans are still capable of handling such situation but agents need to be programmed very particularly about such situations. For example, humans know zero degree angle is equal to 360. We know 180 degree angle is equal to -180 degrees. But agents need to be programmed to handle these situations. Let’s says angle changes from 0 to 720, agent will fail if it’s not taught that 360 and its multiples will be equal to zero too.

3x3 Problems have lot of patterns like objects being same horizontally, or vertically or diagonally. Just as a human would recognize them, I programmed my agent to see if certain set of rules are met, then a pattern is established. This is exactly how humans come up with answers.

In certain problems, like 2x2 problem 04, humans can come up with answers by seeing how a perfect symmetry can be established, which takes them to the answer. Visually this is easy but since I took the textual approach, establishing similar pattern in angle value was tough. In some cases, visual reasoning approach is easy.

While establishing correspondence humans tend to see the best match when all or maximum attributes match. An agent can do the same if right set of rules are established. It’s all on the imagination of the agent designer. I believe that my agent is limited by my knowledge. Agent is coded by human, once correctly coded it can do things way faster than us.

It’s always easy to establish how machine can learn from humans but vice-versa is challenging. That said, I still feel that agents can help humans in seeing the problems from a different perspective. As a human I see a diagonal flip, but my agent taught me that its actually the difference between the two transformations, if difference is +180 or -180, then a similar relationship can be established while generating answer.

Agent adds two transformations, to get the resultant transformation, humans can follow the same pattern. Instead of applying one transformation and then followed by another, agent can apply a XOR on the changes and then add them to get the solution. Agent also helps humans learn that if A->B is applied first or A->C is applied first, the solution should be the same. For example, and attribute changes from A->B but but when we handle A->C, we see it remains same. If A->B is applied first and A->C is applied later, we must still hold on to the changed attribute value.