Project 1

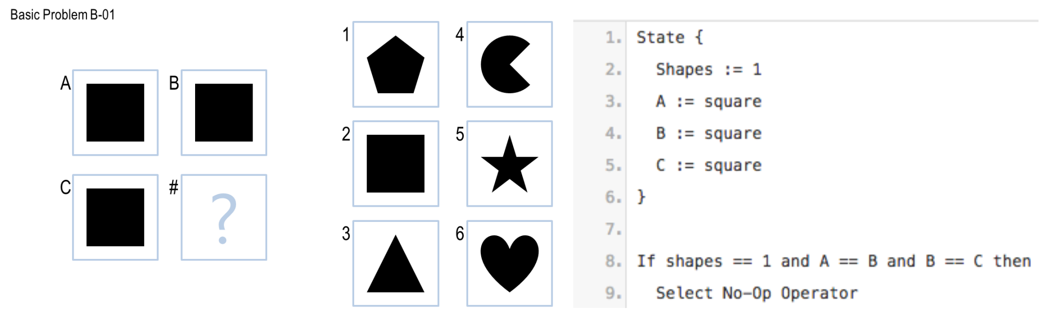
Allan Reyes
reyallan@gatech.edu

Project's Approach

At the beginning, I wanted to tackle this project by building an agent that would be generic enough to attempt solving all the types of problems that will be used throughout the course, i.e. 2x2 matrices as well as 3x3 ones. I quickly realized this was not going to be the easiest way of designing an agent because I immediately got overwhelmed by all the different cases that my agent needed to consider in order to start solving the problems. For example, for 2x2 matrices there are only three given images which are used to find an answer among six possible. On the other hand, 3x3 matrices have eight images and eight answers. Moreover, 3x3 matrices also introduce the notion of diagonal relationships.

Designing an agent that would be able to generically tackle these different characteristics of the problems proved to be more difficult from the beginning than I expected. So, the first modification I did to my approach was separating the two different types of matrices. I decided to focus only on 2x2 matrices as this would be the actual problems that my agent would be asked to solve for this project. By doing so, I reduced the considerations I needed to take because I worked under the aforementioned assumptions of this type of matrices.

My next idea was to solve all problems by constructing a Production System with a working memory that described the characteristics of the problem being solved, and with rules to use that description to decide which operator should be applied. For example, looking at Basic Problem B-01 (*Figure 1.A*), it is obvious that there was no operator applied to the image thus the answer is 2. So, I devised the state and rule shown in *Figure 1.B*.



```
State {
  Shapes := 1
  A := square
  B := square
  C := square
}

If shapes == 1 and A == B and B == C then
  Select No-Op Operator
```

Figure 1. A: Basic Problem B-01. B: Production System derived from the problem.

However, after inspecting the next problems in the set, I realized this approach was not going to work for two reasons. First, clearly there could be more than one shape per frame, and different types of shapes, so the rule of the Production System would need to be too verbose to account for the different types of cases and I believe it would still not have been generic enough. Second, I realized that this approach was not going to scale: I found it challenging to extract enough characteristics of the problems to be used as part of the working memory and then building rules on top of those to differentiate the operators to use to find the answers.

My third idea was to build Semantic Networks based on the shapes inside the figures and their corresponding relationships with the other shapes, as seen in the lectures. Since I decided to use visual inputs only, I knew this was going to be challenging given problems like correspondence, i.e. which shape corresponds to what other shape, etc. So, I took a step back and looked at the problem set from a different angle: all the problems could be solved by manipulating the complete image frame in some way. For example, in Basic Problem B-04 (*Figure 2*), the transformation between frame A and frame B is mirroring the image. This meant that I could find the answer by using affine transformations of the images without really *understanding* what was going on with the image contents. Although this was not going to be a knowledge-based agent, I knew it would be able to solve the majority of the problems.

Nevertheless, because I decided to tackle as many Challenge problems as possible as well, I did end up creating Semantic Networks for problems where affine transformations were not giving the correct results.

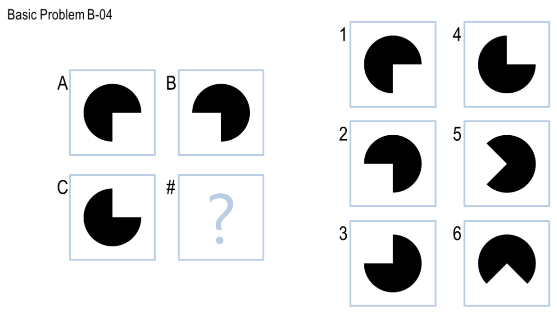


Figure 2. Basic Problem B-04 solvable mirroring the image along the x-axis.

Final Agent: How it Works

My final agent works in two phases, each leveraging the Generate & Test technique. First, it attempts to solve the problem visually via affine transformations of the images without any knowledge representation. It has a list of the different transformations (*Table 1*) it can try and applies them all to the given problem for each of the different *axes*, i.e. row-wise and column-wise, to generate the expected answer. It then iterates through each of the available candidates to test which one best matches the expected answer.

Table 1. List of all transformations known to the agent.

Affine Transformation	Effect
No-Op	Returns a copy of the given image
Mirror	Flips image horizontally (left to right)
Flip	Flips image vertically (top to bottom)
Shape Fill	Fills all shapes inside image
Rotation	Rotates image clockwise
XOR	Finds the difference between images

To find this best match, it uses the Normalized Root Mean Square Error as a similarity measure based on the pixel-by-pixel difference between the images, producing a score between 0.0 (no match) and 1.0 (perfect match). However, my agent only *trusts* answers with a similarity measure of 0.9 or more. Once it has found all the answers based on each of the transformations and axes, it simply takes the one with the highest score. To exemplify this process, for Basic Problem B-05 (*Figure 3*), the highest-scoring transformation would be flipping the image column-wise which results in the correct answer number 4.

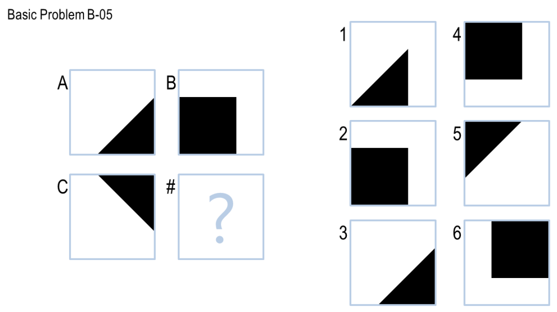


Figure 3. Basic Problem B-05 solvable by applying the flip transformation.

Now, if the visual-only phase does not yield an answer, then my agent attempts to solve the problem semantically via Semantic Networks. In order to build semantic relationships, my agent performs two crucial pre-processing steps: shape extraction and shape classification. To extract shapes, I implemented the contour-following algorithm as proposed by Jonghoon, et al. (2016) that considers a lot of different cases, in particular corners, in order to extract close to 100 percent of the points that form the contour of an object. Having the contour of each of the shapes inside a frame, other attributes are computed like its approximate area, size, whether it is filled or not, etc.

On the other hand, to classify shapes, I attempted two different approaches. I, first, implemented the Ramer-Douglas-Peucker algorithm (Ramer-Douglas-Peucker algorithm, n.d.) to obtain an approximation of the shape's contour. This approximation would yield the number of vertices which were used to determine if the shape was a square, a triangle, etc. However, this method proved to be too unstable resulting in many faulty classifications of images, especially with small shapes whose approximation had low accuracy, which led the agent to answer incorrectly.

In order to improve the classification, I then implemented the $1 Recognizer, as proposed by Wobbrock, Wilson and Li (2007). This is a template-based algorithm that is significantly more accurate since it uses previously recorded shapes as templates to match a given shape to. It is also rotation, translation and scale invariant. However, for this algorithm to work, I had to extract shapes from the problem set and save their contours as templates.

Once all the shapes from each of the frames of the problem have been extracted, my agent then starts generating semantics between them to represent knowledge. As with the first phase, my agent also has a list of relationships (*Table 2*) it can attempt to build for a particular problem. Because this whole process can be very computationally expensive, my agent tries to build one relationship at a time row-wise and column-wise, and then tests each answer to see which one complies with the expected semantics. Moreover, because there is no image similarity measure here, the agent only trusts the answer if it is the *same* for both row-wise and column-wise relationships. If the answers differ, then it continues to attempt the next relationship until an answer is found or all relationships have been exhausted.

Table 2. List of all semantic relationships known to the agent.

Relationship	Semantics
Add-Keep-Delete	Captures addition, deletion or conservation of shapes in frames
Sides Arithmetic	Captures mathematical relationships in frames based on shapes' sides

As an example, Challenge Problem B-03 (*Figure 4.A*) can be solved using the *Add-Keep-Delete* relationship by removing the non-filled squares thus arriving at the fully black image 3. On the other hand, the *Sides Arithmetic* relationship can be used to solve Challenge Problem B-08 (*Figure 4.B*) since it identifies that between the shape in A and B the number of sides was increased by one, and thus C and answer 4 comply with this expectation.

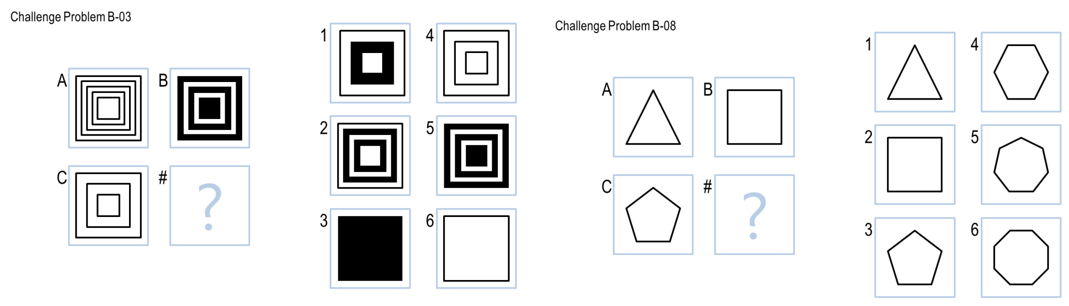


Figure 4. A: Challenge Problem B-03 solvable using *Add-Keep-Delete* semantics. B: Challenge Problem B-08 solvable using *Sides Arithmetic* semantics.

Final Agent: Performance

I believe my agent's performance in this project was good with an overall accuracy of ~92% (44 out of 48 problems). It was able to solve all 12 Basic problems, 11 Test problems, 10 Ravens problems and 11 Challenge problems. I think it is also efficient: the agent's execution time was ~42 seconds meaning an average of ~875 milliseconds per test which can be attributed to the fact that it only attempts to solve problems semantically, which is more computationally expensive, whenever the transformation-based approach does not yield an answer. This saved a lot of time for those problems where knowledge representation is clearly not needed.

In terms of generality, my agent did not overfit. It was able to solve all the problems it was exposed to as part of its *training*, but also around 90% of the *hidden* sets (Test and Ravens), which indicates that my agent is able to apply the list of transformations and relationships that are part of its *toolkit* to problems it has not seen before successfully.

Finally, its performance on the Basic and Test sets is about the same. However, there was only one Test problem that my agent was not able to solve with neither of its phases. Interestingly enough, this problem was not answered incorrectly but skipped. This implies that my agent was finding some answers to the problem, but it was not *confident* enough they were correct, i.e. phase 1 might have found answers whose similarity measure was less than 0.90 or phase two, relationships that did not match row-wise and column-wise. As an improvement for future work, the similarity measure could potentially be lowered.

Final Agent: Limitations

My agent's major limitations are related to the following three aspects: shape extraction, shape classification, and known transformations and semantics. Because my agent uses contour tracing to extract shapes from a figure, it currently fails for those images where there are overlapping shapes (*Figure 5*). Since the algorithm follows black pixels to determine the contour, overlapping shapes would cause it to extract the contour of both shapes together instead of correctly identifying two different shapes. This limitation would lead my agent to incorrectly solve any kind of problems where overlapping is a fundamental part of the transformations or semantics.

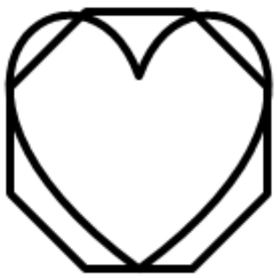


Figure 5. Example of overlapping shapes which the agent would fail to extract separately.

On the other hand, with respect to classification, my agent uses a template-based algorithm which means it is limited by the number of shapes it knows. This will result in incorrect decisions for the kind of problems where the agent encounters shapes that is has never seen before because it will try to classify them as one of the shapes in its library of templates. If the problem needs attributes of this shape, like its number of sides, then it will use incorrect information. This limitation can be overcome by having a *default* shape where all shapes with low matching scores are classified as. At the very least, the agent would not think it is seeing a square when it is not!

Finally, my agent works based on a fixed amount of visual transformations and semantic relationships. Because it does not derive any *new* ones on its own based on the problems it is currently solving, this is its biggest limitation as it will fail to solve any new problems whose characteristics do not fall into what it currently knows. For example, it incorrectly answered Challenge Problem B-10 (*Figure 6*) because the agent does not have a transformation or semantic relationship that considers the *position* of the shapes, which is a critical factor to differentiate between answer 1 (my agent's incorrect choice) and answer 4 (the correct one).

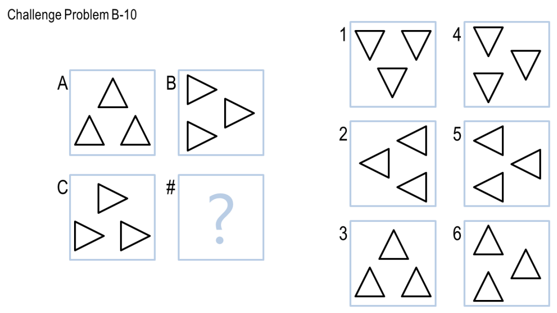


Figure 6. Challenge Problem B-06 where agent chooses incorrect answer (1) due to subtle difference in positioning with correct answer (4).

Human Cognition Connection

Humans learn from experiences in very different ways which means we do not solve the same problem in the same way. However, I think my agent reflects the way *I* learned to solve 2x2 Raven's Progressive Matrices problems in this project.

When I started the project, I went through each problem and annotated each one with the transformation I believed could be applied to each image to generate the correct answer. This is reflected in my revisions as I started providing this reasoning to my agent by implementing each transformation incrementally and tackling similar problems at a time, improving its performance from revision to revision.

Later on, when I started looking at the Challenge problems and found that image transformations where not enough and more sophisticated knowledge representations where needed; I translated this into my agent by empowering it with a semantic solver. Moreover, as with the transformations, I reflected on which relationships could solve the problems and then translated them into my agent's library of available semantics, which is again reflected in the latest revisions of the agent.

With respect to my agent's connection with human processes, I believe it has both similarities and differences. On the one hand, I feel my agent does solve the problems similar to how a human would do so: by applying its past experiences, represented as a list of available transformations and relationships, to solve each new problem encountered, which is a very common approach for reasoning in humans.

On the other hand, as a human, when I was going to the problems, whenever I encountered a problem where any of the transformations or relationships I had been applying did not fit the characteristics of the problem, I would then reflect and *invent* a *new* process that would lead to solve that particular problem, adding it to both my own and my agent's library. In that case, my agent is different from us humans because it is does not have that capability for *invention*. It cannot come up with *new* ways to solve a problem it has never seen before, and it is simply limited by the processes it knows it can try.

References

1. Jonghoon, S., et al. (2016). Fast Contour-Tracing Algorithm Based on a Pixel-Following Method for Image Sensors. In *Sensors*. Basel, Switzerland.
2. Wobbrock, J. O., Wilson, A.D., Li, Y. (2007). Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology.* Newport, Rhode Island, USA.
3. Ramer-Douglas-Peucker algorithm. (n.d.) In *Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm