Project 3

Allan Reyes
reyallan@gatech.edu

Project's Approach

For this project I decided to take the lessons learned from Project 2 (where I wasted a whole week trying to implement a fractal-based algorithm with mediocre results) and stick to my original approach which had great success on the past two projects. Furthermore, in Project 2 I had already worked on extending it to handle 3x3 problems, so applying it to this last project was straightforward. At a high-level my approach involves using visual transformations to manipulate images, as well as, semantic relationships whenever image transformations are not enough to solve the problems.

Once I started following the approach mentioned above, I had two key realizations based on the problems in each of the two sets. First, for set D, I realized that almost all the problems involved finding a *missing* element of the matrix: shapes or patterns or both. For example, Basic Problem D-02 (*Figure 1.A*) involves finding the missing shape, while Basic Problem D-07 (*Figure 1.B*) involves finding the missing shape and the missing pattern around that shape. This allowed me to focus most of my efforts efficiently in developing common heuristics for *finding* shapes and patterns, and then simply adapting them to each of the problems' particular characteristics.

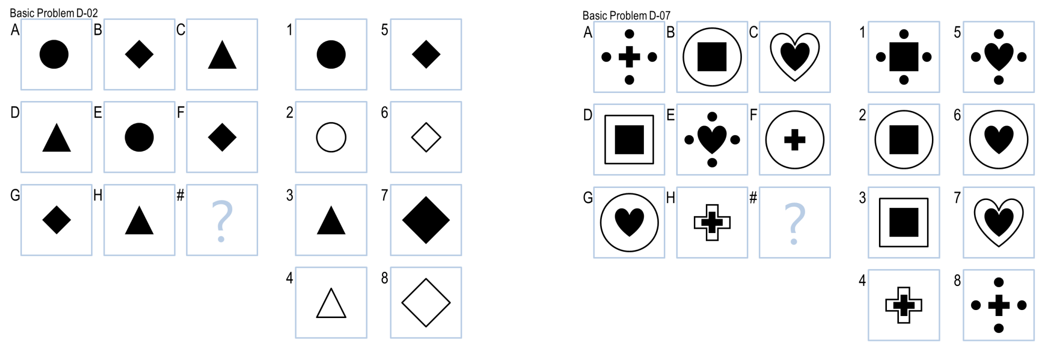
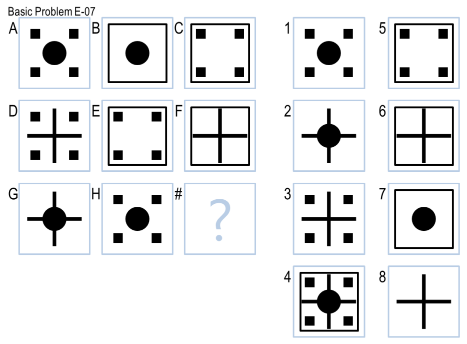


Figure 1. A. Basic Problem D-02 where a shape is missing. B. Basic Problem D-07 where a shape and a pattern are missing

On the other hand, for set E, I realized that the vast majority of the problems could be solved by simple purely-visual transformations. I thought this was interesting because, being the last set, I had imagined the problems in set E would be the hardest ones to solve; however, my agent was able to perform really well on this set because the transformations were fairly simple. As an example, look at Basic Problem E-07 (*Figure 2.A*), at first sight it seems this problem is quite complicated with several different shapes being transformed; however, it can be easily solved by taking the pixel-wise difference between the images in the first two frames (something known as the *XOR* operator). Likewise, Basic Problem E-11 (*Figure 2.B*), despite having different patterns based on filled vs. empty squares, can be solved by finding the common pixels between the images in the first two frames, i.e. the *intersection*.



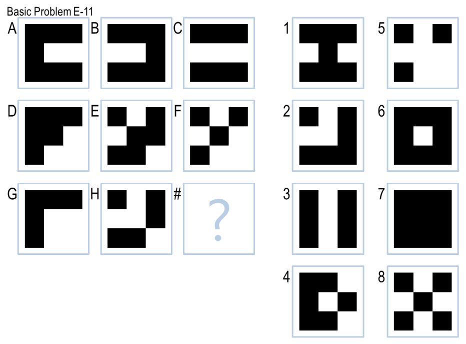


Figure 2. A. Basic Problem E-07 solved by using XOR. B. Basic Problem E-11 solved by using intersection.

One of the other things that I noticed while working on the problems was that the existing transformations and semantic relationships, which were implemented as part of Project 2 for the set C, were causing a lot of false positives when they were running against the new sets D and E. One way of solving this would have been to simply separate the problem sets; however, to me it seemed like a band-aid instead of a solution to the actual problem. After some debugging, I came to the conclusion that the root cause was the fact that existing heuristics were *always* trying to find an answer for the problem even if the problem did not really meet the *conditions*. So, one of the important changes that I made to my approach was adding validations to each heuristic, i.e. verifying against the complete rows or columns that the third frame could be produced by said heuristic, to assert that the problems met certain *criteria* meaning they could potentially be solved by a particular heuristic.

Finally, adding these validations made me realize that applying transformations or semantics to the diagonals of 3x3 matrices was not really worth it because most of the problems exhibited the same patterns in the rows or columns, but not in the diagonals, so having that extra check was causing some false negatives. Thus, I removed the use of diagonals altogether.

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. Note that the transformations that were implemented in Project 2 but did not apply to the problems in set D and E were omitted in *Table* 1 for brevity.

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

Affine Transformation	Effect
No-Op	Returns a copy of the given image
XOR	Finds the difference between two images
Union	Joins two images together
Rotation(90), Rotation(180)	Rotates the image by a certain degree
Centered XOR With Offset	Finds the difference between two images by first offsetting the second image and then centering the result
Image Segment Top Bottom Union	Segments two images in half vertically and merges the top segment of the first one with the bottom segment of the second one
Intersection	Finds common elements between two 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 minimum similarity measure of 0.8 or 0.9 (depending on the transformation) 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 E-04 (*Figure 3*) the highest-scoring transformation would be offsetting the second image to the left, then applying the XOR operator with the first image and then centering the result to produce the correct answer number 8. This procedure applies row-wise, but if we look at the problem column-wise, the only difference is that the second image would be offset to the bottom, and that would produce answer number 8 again.

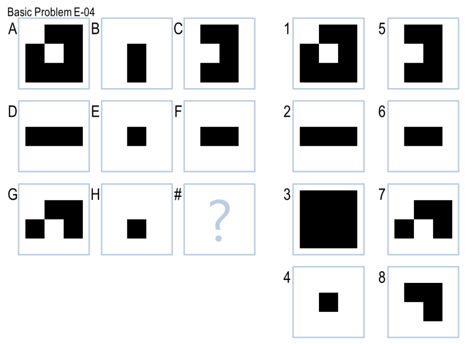


Figure 3. Basic Problem E-04 solved by "Centered XOR with Offset" transformation.

Now, if the visual-only phase does not yield an answer, then my agent goes into the second phase and 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 an image, other attributes are computed like its approximate area, size, whether it is filled or not, etc.

In my current design, the shape extractor merges similar shapes that are too close to each other to handle the fact that the contour-following algorithm will find two contours in shapes with thick outlines (one external and one internal). However, this caused some false negatives in some of the problems; to mitigate this, I also implemented the Connected Components algorithm (n.d.) with the idea of using it as a way of extracting only *bounding boxes* for simple purposes like counting shapes which does not require the handling of two closely-related contours.

On the other hand, to classify shapes, I 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 sets and save their contours as templates.

Once all the shapes from each of the images 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 axes. If the answers differ, then it continues to attempt the next relationship until an answer is found or all relationships have been exhausted. Note again that *Table 2* only shows the semantic relationships that applied to the problems for this project.

As I mentioned previously, a lot of the problems in set D involved finding *missing* shapes and patterns. This property allowed to nicely apply the technique of **Problem Reduction** and let my agent solve sub-problems independently which allowed to find partial solutions by only looking at certain objects inside each image instead of dealing with all the complete image.

As an example of this, Basic Problem D-07 (*Figure 4*) can be sub-divided into two problems: (1) finding the missing center shape and (2) finding the missing pattern that is applied to that shape. As **background knowledge**, the agent knows that the possible patterns are (a) having the center shape surrounded by four other shapes, (b) having the center shape inside another larger shape and (c) have the center shape inside the same shape but larger. Focusing only on sub-problem (1), it is easy to see that the common center shapes are a cross, a square and a heart;

thus, the missing shape is a *square* since there are only two of those when there should be three. Then, focusing on sub-problem (2), it is simple to find that the missing pattern is (a) since there are only two instances of that pattern. When merging the solutions to these two sub-problems, the agent generates the correct answer number 1.

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

Relationship	Semantics	Example Problems
Find Missing Frame	Finds missing frame in matrix	Basic D-02, D-03
Find & Merge Common Shapes Row Column	Finds common shapes in rows and in columns, merging them	Basic D-04, D-05
Find Missing Center Shape & Apply Pattern	Looks for missing center shape in the matrix and then applies a specific pattern to it	Basic D-06
Find Missing Center Shape & Missing Pattern	Looks for missing center shape and missing pattern, out of a known set, in the matrix	Basic D-07, D-08
Find Missing Image Pattern	Looks for missing image pattern that should be valid in groups of three for each image	Basic D-09
Find Missing Shape & Count	Looks for missing shape and its count in the matrix	Basic D-12
Delete Common Shapes & Keep Center Shape	Deletes common shapes between two images, keeping center shape	Basic E-06
Partition Shape into Smaller Shapes with Same Rotation	Partitions shape into smaller shapes keeping rotation	Challenge D-10
Shapes Arithmetic	Performs some arithmetic operation between frames based on the shapes' sides	Challenge E-05, E-06

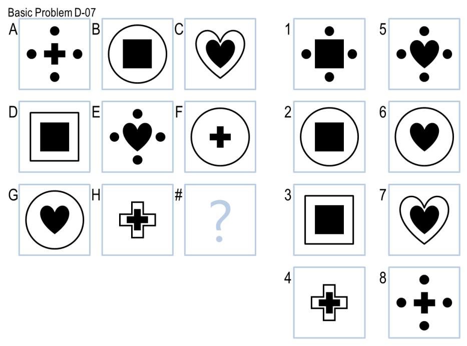


Figure 4. Basic Problem D-07 solved via Problem Reduction.

Finally, in terms of **knowledge representation**, each of the semantic relationships is represented as a **class** with *generate* and *test* methods that construct the expected result (anything from another image to a single number like a count) and test the expected result against every answer to find the correct one (if any), respectively.

Final Agent: Performance

I believe my agent's performance in this project was decent with an accuracy of ~62% (60 out of 96 problems). These results, however, are slightly worse than those from Project 2 (~67% accuracy), and significantly worse from Project 1 (~94% accuracy). Over all the problem sets my agent obtained a **final accuracy of ~69%** (133 out of 192 problems) which I find very satisfying! The complete breakdown of correct answers is shown in *Table 3*.

Table 3. Results from all problem sets

Problem Set	Basic	Test	Ravens	Challenge	Total
B	12	12	11	10	45
C	11	7	4	6	28
D	11	7	2	6	26
E	12	11	6	5	34

I think my agent is not efficient: the agent's execution time was ~25 minutes, with the majority of the time spent solving problems in the sets C and D. This can be attributed to the fact that the majority of problems in those sets cannot be solved using visual transformations, and the agent had to resort to using a long list of semantics which had been accumulating since Project 2, each requiring much more processing when performing tasks like shape extraction. Moreover, all these problems involve 3x3 matrices which are inherently more computationally expensive to handle since they contain more images and more answers which need to be processed in different ways.

More specifically, using **Big O** notation and focusing on the **worst case** where semantic relationships need to be used, both the shape extraction and the shape classification use algorithms that traverse through an image in a row by column fashion which makes them both **O(n^2)**. Moreover, the agent loops through each semantic relationship and loops through each axis (row-wise and column-wise), i.e. produces two nested loops, yielding again **O(n^2)**. Thus. my agent's worst-case runtime efficiency is **O(n^4)**.

In terms of generality, I believe my agent suffered from overfitting. It was able to solve all but one of the problems it was exposed to as part of its *training*, but only around 54% of the *hidden* problems (Test and Ravens), which indicates that my agent was not able to apply the list of transformations and relationships that are part of its *toolkit* as successfully and consistently to problems it had not seen before. It is also important to note the poor performance in the Ravens problems, especially for set D, (only 2 out of 12!) because it hints to the fact that my agent is not successfully solving the *real* Raven's test, indicating a lack of *true* generality.

On the other hand, its performance on the Basic and Test sets is roughly the same with four Test problems being answered incorrectly and two Test problems being skipped. This implies that my agent was finding some answers to those problems, but it was not *confident* enough they were correct, i.e. phase 1 might have found answers whose similarity measure was less than the threshold 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 changed to something different than RMSE, or the validations in the semantic relationships could be relaxed to avoid overfitting too much the requirements needed for each problem.

Final Agent: Limitations

My agent's major limitations are related to the following three aspects: shape extraction, runtime execution, 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. Due to this limitation my agent will fail to solve problems containing this type of shapes like Basic Problem D-10.

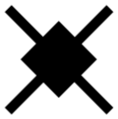


Figure 5. An image with overlapping shapes: a diamond and a cross.

On the other hand, my agent is not efficient, and its runtime execution became a limitation during its development since it slowed down the iteration process of making changes and validating them, having to wait almost 30 minutes each time. Moreover, this execution will keep growing exponentially either if the transformations and semantics increase in complexity or if my agent were to be presented with more problems. As an improvement, optimization techniques, like caching, will need to be explored to speed up the computation of certain transformations.

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 was not able to answer Challenge Problem E-07 because the agent does not have a transformation or semantic relationship that considers the *vertices* of shapes and performs *operations* on them.

Human Cognition Connection

I think my agent reflects the way *I* learned to solve 3x3 Raven's Progressive Matrices problems in this 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 problems that my agent was not solving because image transformations were not enough, I knew more sophisticated knowledge representations were needed. Thus, 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 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. In particular, my agent uses the **Generate and Test** technique to produce expected results and evaluate each answer against them; likewise, my agent also exhibits the **Problem Reduction** methodology, where it divides the problem into smaller tasks like finding a particular shape or a particular pattern, and then combines the solutions to produce the expected result. These two techniques are central to how us humans solve day-to-day problems.

On the other hand, as a human, when I was going through the problems, whenever I encountered one 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. Connected-component labeling (n.d.) In *Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/Connected-component_labeling
2. Jonghoon, S., et al. (2016). Fast Contour-Tracing Algorithm Based on a Pixel-Following Method for Image Sensors. In *Sensors*. Basel, Switzerland.
3. 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.