Project 2

Allan Reyes
reyallan@gatech.edu

Project's Approach

For this project, I decided to try a new approach with hopes of it being general enough to easily extend to the sets for the next project as well. With that in mind, I implemented the Confident Ravens Algorithm as proposed by McGreggor and Goel (2014). A crucial part of this algorithm is the fractal encoding process whose objective, at a high level, is to generate features that can describe the transformation between two or three images via fractals: a set of elements that characterize the conversion between a particular "block" of a source image into a "block" of a destination image.

The implementation of the fractal encoding algorithm proved to be more difficult that I had anticipated. Specifically, I believe the way it was described in the paper was not intended to be reproduced. I turned to McGreggor's actual dissertation thesis, "Fractal Reasoning" (2013), hoping the process would be explained with enough details to be reproducible. Fortunately, though there were still a couple of open questions, the author dove deep into the inner workings of the algorithm which provided enough detail to understand the process better.

I was able to implement a working fractal encoding algorithm, at least to the extent of my understanding, and verified I could actually reconstruct a destination image given the encoding from the source image. With this algorithm in place, I proceeded to develop the actual Confident Ravens Algorithm, whose pseudocode was easy to follow and reproduce. The results published in the paper were astounding, so I was hoping to get similar or near-similar results when running it against the problems for this project.

Unfortunately, my results were significantly worse than those published in the paper. In fact, my implementation was only able to solve two problems of the Basic Set C. Moreover, the running time of the algorithm was exponentially large, taking almost 30 minutes to go through the set of Basic problems only. I did not even bother running it against the Test set since I knew the auto-grader would time out. Given the poor results and the terrible performance, I decided to abandon this approach and the idea of implementing something completely new.

So, I decided to stick with the same approach as Project 1 since it had given really good results, and work on extending it to handle the 3x3 problems for this project. This approach involves using visual transformations to manipulate the images, as well as, semantic relationships whenever image transformations are not enough to solve the problems.

One of the key realizations I had once I started following the approach mentioned above, was that for the problems in this particular set, almost all middle frames are simply, what I call, *transition* frames. In other words, they seem to be only *middle* states to an actual final transformation; which means that, for the most part, these frames can be ignored and only the best transformation between the first and third frame needs to be found. As an example, look at Problem C-09 (*Figure 1.A*). It is clear that the shapes from the first frame, image "A" for example, are switching sides. The middle frames, like image "B", are simply the *transition* between the shapes switching from one side to the other, and thus the agent only needs to care about moving the shapes from the first frame to the third, which is the actual final transformation. Another example of this property can be seen in Problem C-07 (*Figure 1.B*) where the circle in the first frame is moving to the other side in the third frame; the second frame, is simply the natural moving transition.

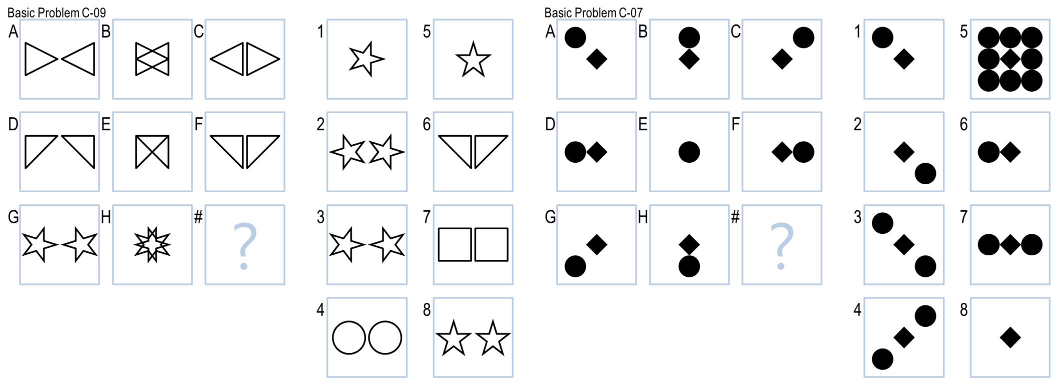

Figure 1. A: Basic Problem C-09 with middle transition frames. B: Basic Problem C-07 with middle transition frames.

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, column-wise and diagonal-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)
Image Duplication	Duplicates the image a certain number of times either overlapping, non-overlapping or side by side
Image Switch Sides Horizontally	Switches two sides of an image
Image Top Down Segmentation	Segments and deletes parts of image from top to bottom
Rotation and Union	Rotates image and merges it with another

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 0.8 or 0.9 (depending 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, let’s look at one of the most interesting transformations: Image Duplication. I found this transformation to be very useful because it applies to a large number of Basic and Challenge problems (*Table 2*). By looking at any of those problems, it is clear that the image in frame "A" is being duplicated two or three times in the next frames either row-wise or column-wise. The key distinction between all of them are the following characteristics: the number of times the image is duplicated, whether the duplicates overlap, not overlap, or are placed side-by-side, and the axis of the duplication, i.e. either horizontal, vertical or diagonal. The problems solved via this transformation and their characteristics are listed in *Table 2*.

This transformation also ties back to the realization explained previously: the middle frames can be seen as *transitions* and can be ignored. This simplifies seemingly-complex problems which can be correctly solved by just looking at the first and third frames. For example, for Challenge Problem C-07 (*Figure 2*), my agent does not have to deal with the rotation that is happening in the middle frames; it simply realizes that the first image is being duplicated three times without overlap, and that is enough for it to find the correct answer.

Table 2. List of problems solved via Image Duplication

Problem	Times	Mode	Axes
Basic C-03	3	Non-overlapping	Diagonally
Basic C-04	3	Overlapping	Horizontally/Vertically
Basic C-06	3	Side-by-side	Horizontally/Vertically
Basic C-10	2	Non-overlapping	Horizontally/Vertically
Challenge C-03	3	Non-overlapping	Horizontally/Vertically
Challenge C-06	3	Non-overlapping	Vertically
Challenge C-07	3	Non-overlapping	Horizontally/Vertically
Challenge C-11	3	Non-overlapping	Vertically

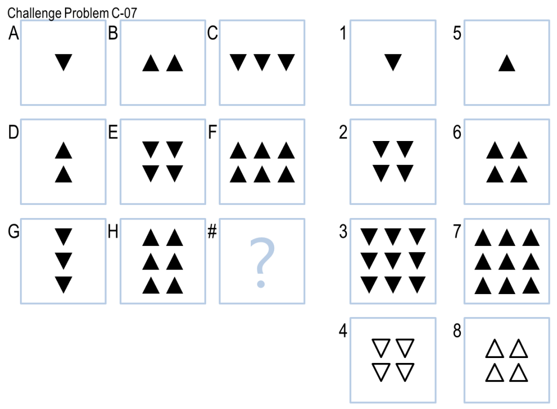

Figure 2. Challenge Problem C-07 where the middle rotation can be ignored.

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 a frame, other attributes are computed like its approximate area, size, whether it is filled or not, etc.

I also implemented the Connected Components algorithm (n.d.) with the same purpose of identifying shapes with the idea of using it as a way of extracting shapes the contour-following algorithm might have missed. However, in practice, none of the algorithms provided any extra improvement of performance over the other, so I stuck with the contour-following process since that had also proven to yield good results since Project 1.

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 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 3*) 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, column-wise and diagonal-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 all three axes. If the answers differ, then it continues to attempt the next relationship until an answer is found or all relationships have been exhausted.

Table 3. List of all semantic relationships known to the agent.

Relationship	Semantics
Shape Scaling	Looks at the difference between the scale of shapes from frame to frame
Shape Fill Points System	Generates a points system based on the 'filled' attribute of shapes, e.g., filled shapes are worth 2 points while empty ones, 1 point
Inverted Diagonal Union	Generates a semantic relationship where image "C" is always merged with image "G"

As an example, Basic Problem C-02 (*Figure 3.A*) can be solved using the *Shape Scaling* relationship by noticing the change in scale of the outer square between each frame. On the other hand, the *Shape Fill Points System* relationship can be used to solve Basic Problem C-08 (*Figure 3.B*) by assigning 2 points to the empty squares and 1 point to the filled ones and counting the difference in points between each frame. Finally, Basic Problem C-12 (*Figure 3.C*) can be solved by merging image "C" with image "G" generating the correct answer 8 via the *Inverted Diagonal Union* relationship.

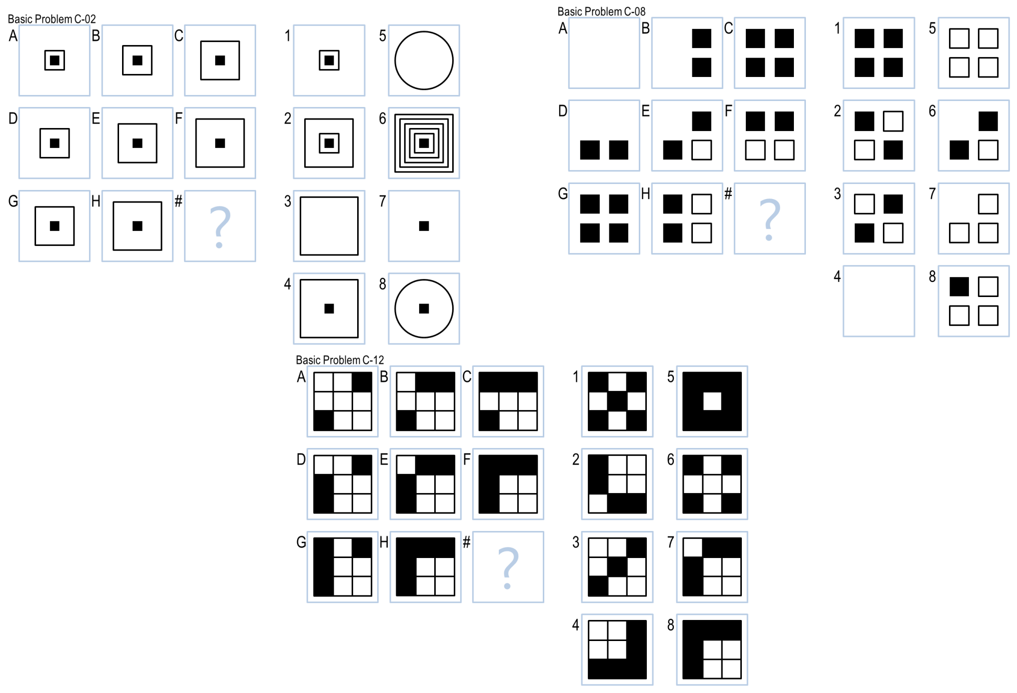

Figure 3. A: Basic Problem C-02. B: Basic Problem C-08. C: Basic Problem C-12

Final Agent: Performance

I believe my agent's performance in this project was decent with an accuracy of ~67% (32 out of 48 problems). These results, however, are significantly worse than those from Project 1, where my agent achieved a ~94% accuracy. Over both the problem sets B and C, my agent currently has an accuracy of ~79% (76 out of 96 problems). The complete breakdown of correct answers is shown in *Table 4*.

Table 4. Results from problem sets B and C.

Problem Set	Basic	Test	Ravens	Challenge	Total
B	12	12	10	11	45
C	12	9	4	7	32

I think my agent is not efficient: the agent's execution time was ~10 minutes, with the majority of the time spent solving problems in the set C (around 9 minutes). This can be attributed to the fact that problems in that set 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; in particular for those problems where visual transformations were not enough, and the agent had to resort to using semantics which require much more processing when extracting shapes.

In terms of generality, I believe my agent suffered from overfitting. It was able to solve all 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 (only 4 out of 12!) because it hints to the fact that my agent is not solving the *real* Raven's test as successfully as I would have hoped for, indicating a lack of *true* generality.

On the other hand, its performance on the Basic and Test sets is roughly the same with two Test problems being answered incorrectly and one Test problem being skipped. This implies that my agent was finding some answers to that 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, column-wise and diagonal-wise. As an improvement for future work, the similarity measure could potentially be lowered, or semantic relationships could specify whether the answer should be consistent across axes or not, for cases where the semantics might only be valid for one particular axis.

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 empty shapes with thick outlines (*Figure 4*). The algorithm detects two contours, the inner and outer; however, because the outline is so thick, it is not able to correctly mark these two shapes as belonging to the same one since they are separated by a large margin. This limitation would lead my agent to believe there are more shapes in a figure than there truly are, potentially causing it to select the incorrect answer.

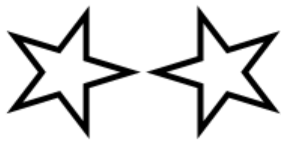

Figure 4. Shapes with thick outlines where my agent identified 4 instead of 2 shapes.

On the other hand, as it was previously mentioned, my agent is not efficient, and I believe its runtime execution will become a major limitation when solving the last two sets of problems. If my agent maintains the same level of runtime performance, then it is expected to take almost 30 minutes to solve all four sets of problems which is very close to the hard limit the agent has. However, if the transformations and semantics in the next sets are more computationally expensive, then my agent will definitely reach that limit. 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 C-09 (*Figure 5*) because the agent does not have a transformation or semantic relationship that considers the *combination* of the different shapes in an image into a single one *ordered* by each shape's *size*.

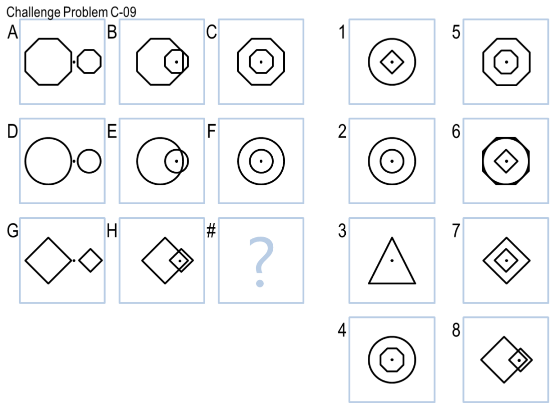

Figure 5. Challenge Problem C-09 where shapes are combined and ordered based on their size.

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 3x3 Raven's Progressive Matrices problems in this project.

Following the same approach, as in Project 1, 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.

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. 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. McGreggor, K. (2013). Fractal Reasoning. Atlanta, Georgia, USA.
4. McGreggor, K., Goel, A. (2014). Confident Reasoning on Raven's Progressive Matrices Tests. *In Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. Québec City, Québec, Canada.
5. 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.