Dustin Jones

Ashok Goel

CS-7637

2016-11-29

Project 3 Reflection

Introduction

Using a series of tests, the agent built for Project 3 attempts to solve Ravens Progressive Matrices (RPMs) by analyzing visual data in a human-like manor. Even though the agent performs on par with most other agents produced through Knowledge-Based Artificial Intelligence, it is still lacking the most fundamental aspect of being an artificial intelligence: the ability to self-learn.

Migration from Project 2 to Project 3

For Project 2, I designed an agent for solving RPMs using a visual approach. Through research, I identified 2 key metrics that were documented (Joyner et al, 2015) to produce the most consistent results for comparing images: the dark pixel ratio (DPR) and intersection pixel ratio (IPR). The DPR is useful for identifying the potential for key characteristics between 2 images such as change (or lack thereof) of shape or fill, as well as allowing transitive comparison across a series of images. The IPR is useful for similar reasons, as well as identifying to what degree 2 images overlap.

The logic for the Project 2 agent was designed around direct numerical analysis of entire images. Images were compared in sets, for example from A to B, and B to C. Directly comparing these values allowed for identification of trends across a series of images, such as darkening, lightening, and increasing or decreasing number of pixels in identical positions. This was successful for Basic Problems C with 12/12 correct, and Test Problems C with 9/12 correct and 3 incorrect. While this was successful for Project 2, the direct numerical comparison did not allow for the sustainable addition of new patterns when applied to the problems for Project 3. The design also left significant room for erroneous results to be returned without a good metric for establishing a level of certainty.

Learning from the design mistakes of Project 2, the agent for Project 3 manages the relationships between images in a much more explicit manor. It does this by taking a list of every known, computable relationship within an RPM, calculating a set of answers that satisfy each case and testing to identify if there is a singular best answer across the results from each case. This set of cases was built incrementally by identifying and naming the types of transformations across rows and columns.

Knowledge Representation and Reasoning

As mentioned previously, the downfall of the Project 2 agent was in the generality of relationships, and not being able to clearly identify one type of relationship from another. When redesigning the agent for Project 3, it was extremely important to be able to appropriately group and define sets of images in a meaningful way. By codifying the core rules for RPMs – that there exists a relationship across rows, and a relationship across columns – and explicitly building frames to encapsulate those rules, the agent is much better equipped to solve new types of problems. While the agent doesn't directly translate the images into verbal representations similar to the provided verbal structures for Project 1 and Project 2, it does define some object features as named properties, such as an object being ‘hollow’ or ‘solid’, or if one object surrounds another without touching, it’s defined as ‘surrounding’.

The agent begins by assigning each `RavensFigure` into 2 `FigureSeries`, based off its respective row and column, building a row `FigureSeries` and column `FigureSeries` for each figure, including each answer figure as shown in Fig. 1. This results in a total of 20 individual `FigureSeries`, with most of the reasoning taking place over the Col1, Col2, Row1, and Row2 because they’re known quantities.

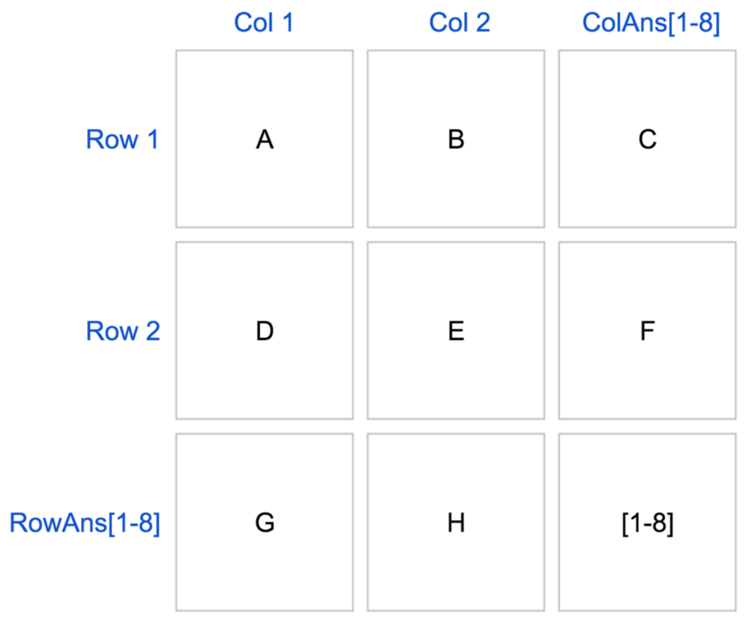

Fig. 1 Assignment of FigureSeries objects

Each `FigureSeries` is comprised of 3 `RavensFigures`, with several functions designed to operate on, and test all 3 as a unit, rather than as individual images. Each `RavensFigure` is further processed into a `VisualFigure` object, where it is transformed from an image into a set of `Pixels` (a simple object consisting of an ‘x’ and a ‘y’ coordinate), and then further divided into distinct sets of disconnected `Pixels`. The process for dividing into sets of disconnected pixels is by far the most time-consuming aspect of the agent, taking 2 orders of magnitude longer to process than any other component, but fortunately only needs to occur once per figure. It is highly likely that there is a more efficient algorithm to accomplish the task, however the agent never failed to complete all the graded problems within the allotted timeframe. Breaking images into sets of disconnected `Pixels` allows the agent to build a library of key characteristics for a figure, allows for more detailed operations within a figure, and more detailed comparisons across figures.

Once the initial processing of figures is complete, the RPM is run through a series of tests; at the time of this writing there exists approximately 20 unique tests. Each test can produce multiple answers that will be compiled later in processing. Every test is designed to identify a specific relationship across a set of `FigureSeries` (either rows or columns), most running twice, once for the horizontal direction, and once for the vertical direction, and each run producing its own set of matching answers.

When building each of these tests, there were a few important guidelines to ensure that they were useful and not needlessly consuming resources:

1. Runs regardless of direction (across rows, and across columns)
2. General enough that if there is a directionality, or ordering, that all variations of those directionalities or orderings are also tested
3. Strict intolerance of 'guessing' to avoid producing false positives, only answers completely satisfying the test conditions are returned
4. Does not alter the original data so that tests can be run in an arbitrary order without impacting any other test

By strictly observing these guidelines, this allowed for a greatly simplified process for selecting an answer. Each test casts votes for the answers that satisfy its conditions, every vote is weighted equally across all tests as there are no incorrect answers, just additional relationships. Tests resulting in either no answers, or all answers are excluded from this voting process as they either do not apply to the problem, or they are too generalized for the problem and will not add value to finding a solution. Once all tests have completed, the resulting votes are tallied and if there is a singular answer with a simple majority that answer is selected. However, if multiple answers are tied for the highest number of non-zero votes the problem is skipped as the tests were insufficient for determining an answer with absolute correctness. While it is not the default behavior of the agent, it is possible to allow for guessing between multiple answers, either by randomized selection, or by weighting the answers returned by specific tests with relation to their history of correctness when encountering similar problems in a diagnostic fashion.

Walkthrough of an RPM

Although there are far too many tests to cover each within the scope of this paper, the general workflow across each is quite similar.

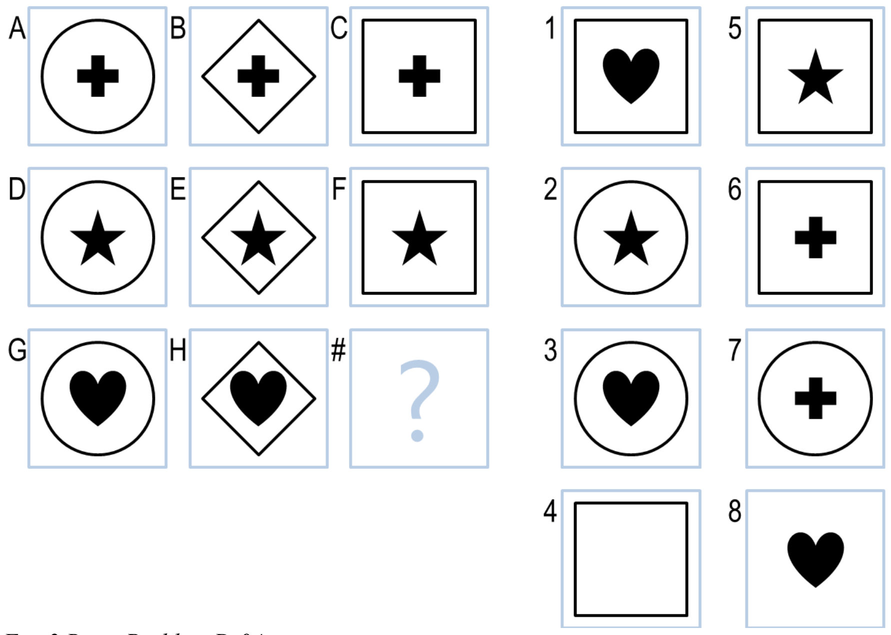

Fig. 2 Basic Problem D-04

As shown in Fig. 2, there are frequently several types of relationships present in an RPM.

- Central filled shapes are consistent across columns, but change across rows
- Outer hollow shapes are consistent across rows, but change across columns

To codify this scenario, the agent examines each of the shapes individually, as sets of disconnected pixels. For both of these relationships, there will be one test that returns results, but run twice, once for the row relationships and once for the column relationships.

When the columnar processing takes place, it will begin by first identifying and then removing all identical shapes with respect to individual columns as shown in Fig. 3, resulting in the removal of all of the matching outer hollow shapes. This eliminates answers 2, 3, 4, 7, and 8 from the test's result set because there is not a matching outer box.

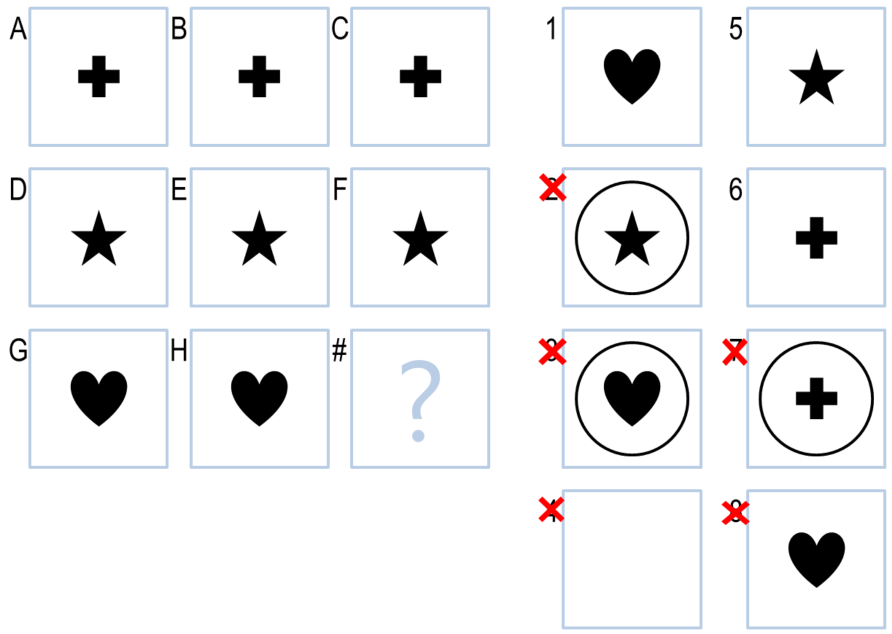

Fig. 3 Basic Problem D-04 - Processed Columns

Once this is complete, a set-based comparison is made, determining that there are no shapes that repeat in a single column, but that there are shapes repeated across all columns with respect to the same figures in each (that is, the 'plus' is always in the first figure, the 'star' is always in the second figure, and the 'heart' is always in the third figure). Identifying that the 'heart' is always in the third figure eliminates figures 5 and 6, leaving only figure 1 as the correct answer.

When row processing takes place, it will begin by first identifying and removing all identical shapes with respect to individual rows as shown in Fig. 4, resulting in the removal of all of the matching inner filled shapes. This eliminates answers 2, 4, 5, 6, and 7 from the test's result set because there is not a matching inner shape.

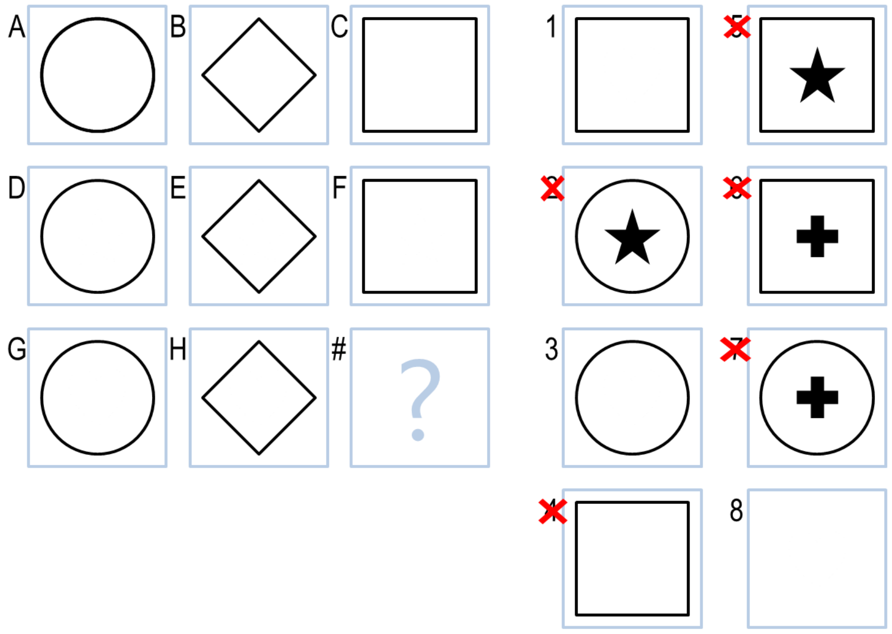

Fig. 4 Basic Problem D-04 - Processed Rows

Once this is complete, a set-based comparison is made, determining that there are no shapes that repeat in a single row, but that there are shapes repeated across all rows with respect to the same figures in each (that is, the 'circle is always in the first figure, the 'diamond is always in the second figure, and the 'square' is always in the third figure). Identifying that the 'square' is always in the third figure eliminates figures 3 and 8, leaving only figure 1 as the correct answer.

After the votes across all tests have returned, figure 1 is ‘elected’ as the best choice because it was the answer that received the most votes. There are several other tests that take place, also identifying figure 1 as a possible answer, but also having identified other figures that fit their respective relationships, however there is not space to cover them all here.

Performance and Mistakes

At the time of this writing the agent is currently able to solve 9/12 problems from the Basic Problem D set with 0 incorrect, and 11/12 problems from the Basic Problem E set with 0 incorrect. When running against the Test Problem D set it answers 5/12 correctly with 0 incorrect, and 9/12 problems from the Test Problem E set with 0 incorrect. While the Test scores aren’t as high as I might have hoped, they do satisfy my primary goal of scoring 0 incorrectly. This effectively demonstrates that my voting technique as well as my calculation of known relationships is entirely successful, makes no mistakes, and the only limiting factor would appear to be interpretation or understanding of the known relationships in the Basic problems.

Because there is no way to build a history of erroneous results and learn from newly encountered problems, the optimal configuration of the agent does not allow for guessing when the answer selection process ties over multiple answers. However, by enabling purely randomized guessing across the tied answers, the agent is able to greatly reduce the probability of guessing incorrectly. When running the agent with randomized guessing enabled, the agent was able to solve 8/12 problems from the Test Problems D set with 4 incorrect, and 10/12 problems from the Test Problem E set with 0 incorrect. This would suggest that additional rules are necessary to solve the D set with absolute correctness, and that the E set is overfitting in some area.

The Basic Problem Sets D and E consistently run in approximately 3.5 minutes on my local machine. The majority of this time is due to the inefficiencies of the disconnected shape algorithm, where that processing for Basic Problem E-11, E-04, D-09, and D-11 account for 2 minutes of that time, with most other problems being solved in under 5 seconds each. While improving this particular algorithm would greatly improve the time efficiency of the agent, it was not a limiting factor in the success of the agent and was left as a problem for another day.

Relating to Knowledge-Based Artificial Intelligence

Unfortunately, there is very little behavior of the agent that can be related directly to KBAI. At its core, the agent is simply a basic set of algorithms utilizing standard object-oriented design methodologies, and insisting that it is anything more would be disingenuous to both the reader and the class. The primary challenge with building an agent that can use some of the techniques discussed in class is that the agent is missing a critical source of information – the feedback loop. As discussed throughout the course, the architecture for a cognitive system, and more specifically the Deliberation cycle is dependent upon 3 processes: reasoning, learning, and memory as show in Fig. 5.

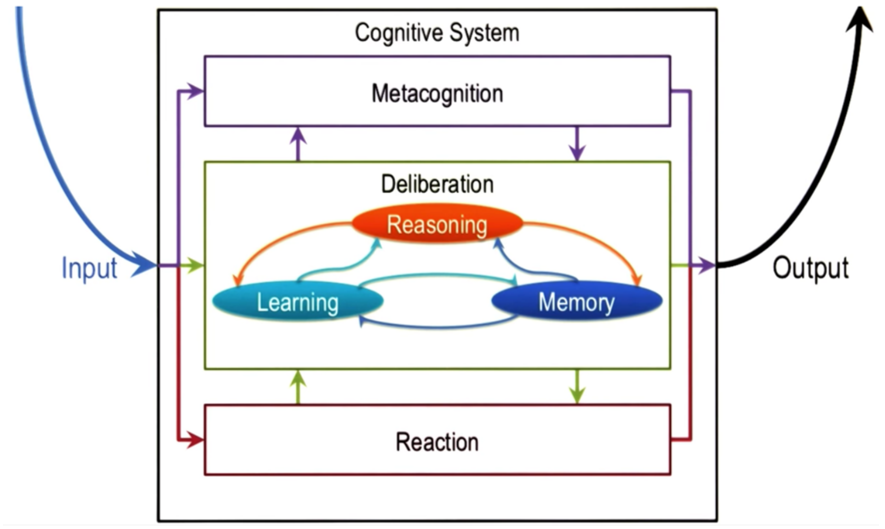

Fig. 5 Udacity - Cognitive System Architecture

While the agent is entirely capable of reasoning over any RPM abiding by the row-column relationship rules, without a mechanism to automatically determine where the reasoning process went wrong, there is no opportunity for the agent to learn. The agent does not receive any indications as to the correctness of a response, nor is it allowed to store information which might be used to build better responses in the future or be used for metacognition to identify rules that perform well or perform poorly. Without the ability to engage in learning or memory activities, the agent is simply left to use its in-built reasoning capabilities.

It is possible however to design the agent in such a way that it can be presented with new problems outside the constraints of the Bonnie-Agent framework. Given unlimited time and resources, one possible way of designing the agent would be to provide a large number of training RPMs where there is a consistent set of observable patterns from which it can build an internal representation, perhaps using a mechanism such as version spaces or a decision tree to be able accurately classify that pattern. The downside to this approach would be that it is much more akin to Machine Learning than it is to Knowledge-Based AI.

Relation to Human Cognition

The overall design of the agent as well as the process for designing the agent is surprisingly similar to how many humans approach solving Ravens Progressive Matrices. By limiting the use of the Dark Pixel Ratio and Intersection Pixel Ratio mainly to simple similarity tests (comparing for similar size, location, shape, and fill) the agent can 'see' similarities the way a human can. Humans also apply a significant amount of background knowledge to the problems they encounter, everything from recognizing a shape as a square vs a triangle, to recognizing basic math problems like addition and subtraction. While the agent currently doesn't have anywhere near the same level of background knowledge and capabilities as a human, the process for building new tests and new ways of recognizing objects directly translates to the pattern observation process seen in humans from day to day life. As new problems are encountered, new tests can be added to solve for those problems in the future.

Adding the ability to make a randomized guess to the agent after sufficiently reducing the total number of answers also mimics human behavior. If the risk/reward balance is sufficiently high, humans will often guess at problems as well.

Works Cited

Joyner, David A., Darren Bedwell, Chris Graham, Warren Lemmon, Oscar Martinez, and Ashok K. Goel. "Using Human Computation to Acquire Novel Methods for Addressing Visual Analogy Problems on Intelligence Tests." *Proceedings of the Sixth International Conference on Computational Creativity*. Provo, Utah. 2015. Web.

Udacity. "Cognitive System Architecture - Georgia Tech - KBAI: Part1." Online video clip. YouTube. YouTube, 23 Feb 2015. Web. <https://www.youtube.com/watch?v=HBLtEMa-61o>