Michael D. Brown
CS 7637 KBAI: Cognitive Systems - Project 1 Reflection
06 June 2015

1: Introduction

For the purposes of this project reflection, I assume that the reader is familiar with the source material for CS7637, the overall project description, and the project one description. My intent is describe the overall design and implementation of the agent, not to summarize the background information my peers and the course instructors are familiar with.

2: Overall Design

I have implemented my AI agent in Java, using a modified Strategy design pattern. The strategy design pattern encapsulates algorithms with a common interface, allowing the application developer to define a family of algorithms that can be selected at runtime.

My agent mimics human problem solving strategies with this pattern. When humans attempt to solve a problem and come up with a suboptimal or failed solution, they attempt to solve the problem another way until they are satisfied (or give up!). The Raven's Progressive Matrices (RPM) vary widely in how they measure human intelligence, and for an AI agent to be successful it must be able approach the same problem in multiple ways or have different strategies to solve different problems.

2.1: Agent Level Problem Solving

For the purposes of completing project one, I focused my development into two problem solving layers. This highest level layer is largely provided by the project skeleton. The agent receives the problem, examines it, and based on the problem metadata (primarily problem dimensions and whether verbal data is available), the agent selects one or more strategies that will best allow it to solve the problem.

These strategies are implemented as independent problem solving algorithms. Each algorithm is invoked using a copy of the problem, and returns an answer with a corresponding confidence level. The agent incorporates a high level test taking strategy where it analyzes the answers available and decides whether to answer or skip the problem based on the relative confidence. The current threshold for skipping a problem is below 14% confidence, derived from the penalty for guessing on 2x2 RPM problems.

2.2: Strategy Level Problem Solving

The second layer is the strategy layer. Given the scope of the first project, I developed one strategy for the agent to utilize. This strategy is only effective for 2x2 RPM problems that have verbal descriptions. The agent will skip any problem not meeting these requirements.

This strategy represents the agent's available knowledge of the problem as a semantic network, and tests the available answers for fitness as a solution. It uses elements of problem reduction and frames to facilitate reasoning, and relies on the structure of the problem to generate the solutions to be tested.

Details regarding the implementation of this strategy are covered in section three of this paper. Further discussion of how the agent represents knowledge, reasons about the solution, and selects an answer will be with specific respect to this strategy (hereafter dubbed the 2x2SNGTVerbal strategy).

3: Agent Implementation

This section details the specific implementation of the agent to provide the necessary background for further discussion of the agent's capabilities.

3.1: Support Data Structures

The agent utilizes several support structures for consolidation of problem specific information. The first is RavensSolution, which serves as the common return interface for a strategy. Each strategy invoked by the agent returns a RavensSolution object as a response allowing the agent to decide which strategy has the best answer. A RavensSolution object has two components:

- Integer value containing the best answer the strategy can determine
- Double value indicating the strategy's confidence in the answer. The confidence level is interpreted as a percentage at or below 100%.

Additionally, two simple structures used for determining figure and object correspondence are implemented, FigureCorrespondence and ObjectCorrespondence. These structures represent the similarity between two elements (RavensFigures or RavensObjects) as an integer value.

3.2: Static Utility Classes

Two utility classes containing frequently re-used code are implemented to maintain readability in the code base. They are TransformationUtilities and CorrespondenceUtilities. They contain functions to determine if an object is unchanged, rotated, reflected, etc. between figures and which objects correspond to each other across figures.

3.3: Transformation Classes

The provided project skeleton provides the low level frames that form the basis for this strategy's semantic network. The 2x2SNGTVerbal strategy extends the knowledge representation with the implementation of ObjectTransformation and FigureTransformation.

3.3.1: ObjectTransformation

ObjectTransformation represents the transformation between two RavensObjects, based partially on the spatial orientation of the figures (horizontal vs. vertical). At instantiation, the constructor determines the applicable transformations and stores them as a comma separated string.

ObjectTransformation objects support a comparison function that returns a double value interpreted as a percentage that represents the similarity between two ObjectTransformation objects. This is calculated by examining the transformations between each set of objects and calculating a simple average of matching values.

3.3.1: FigureTransformation

FigureTransformation objects represent the transformation between two RavensFigures based partially on the spatial orientation of the figures and a hash map of figure "nesting". Nesting refers to the level of containment of an object, for example:

The filled square is contained by two other objects, and has a nesting value of 2. The unfilled square has a nesting value of 1, and the circle has a value of 0.

FigureTransformation objects support a comparison function that returns a double value interpreted as a percentage that represents the similarity between two FigureTransformation objects. This is calculated by examining the ObjectTransformations between each set of objects and calculating a simple average across the entire figure.

3.4: Strategy – 2x2SNGTVerbal

The strategy this basic AI agent is built upon is designed to solve 2x2 RPM with verbal descriptions. It implements RavensStrategy, an abstract class that defines the common strategy interface for the agent. It primarily implements the method solve(), which returns a true Boolean value if a solution was successfully found. The RavensSolution object is retrievable after the call to solve.

4: 2x2SNGTVerbal Strategy Implementation

4.1: Problem Solving Process

4.4.1: Semantic Network Construction

Solving RPM using this strategy relies heavily on the representation of knowledge constructed from the verbal data. This knowledge representation is created when the agent calls the solve method for the strategy. The following steps are taken in order to complete the semantic network:

1. The strategy obtains the figures from the problem hash map
2. The nesting levels for the objects in the problem are determined via utility method
3. Reference FigureTransformation objects are constructed between Figures A:B and A:C.
    a. When FigureTransformation objects are created they:
        i. Determine which objects correspond to each other
        ii. Generate ObjectTransformation objects between associated objects
4. Possible solution FigureTransformation objects are constructed for Figures B:1-6 and C:1-6.

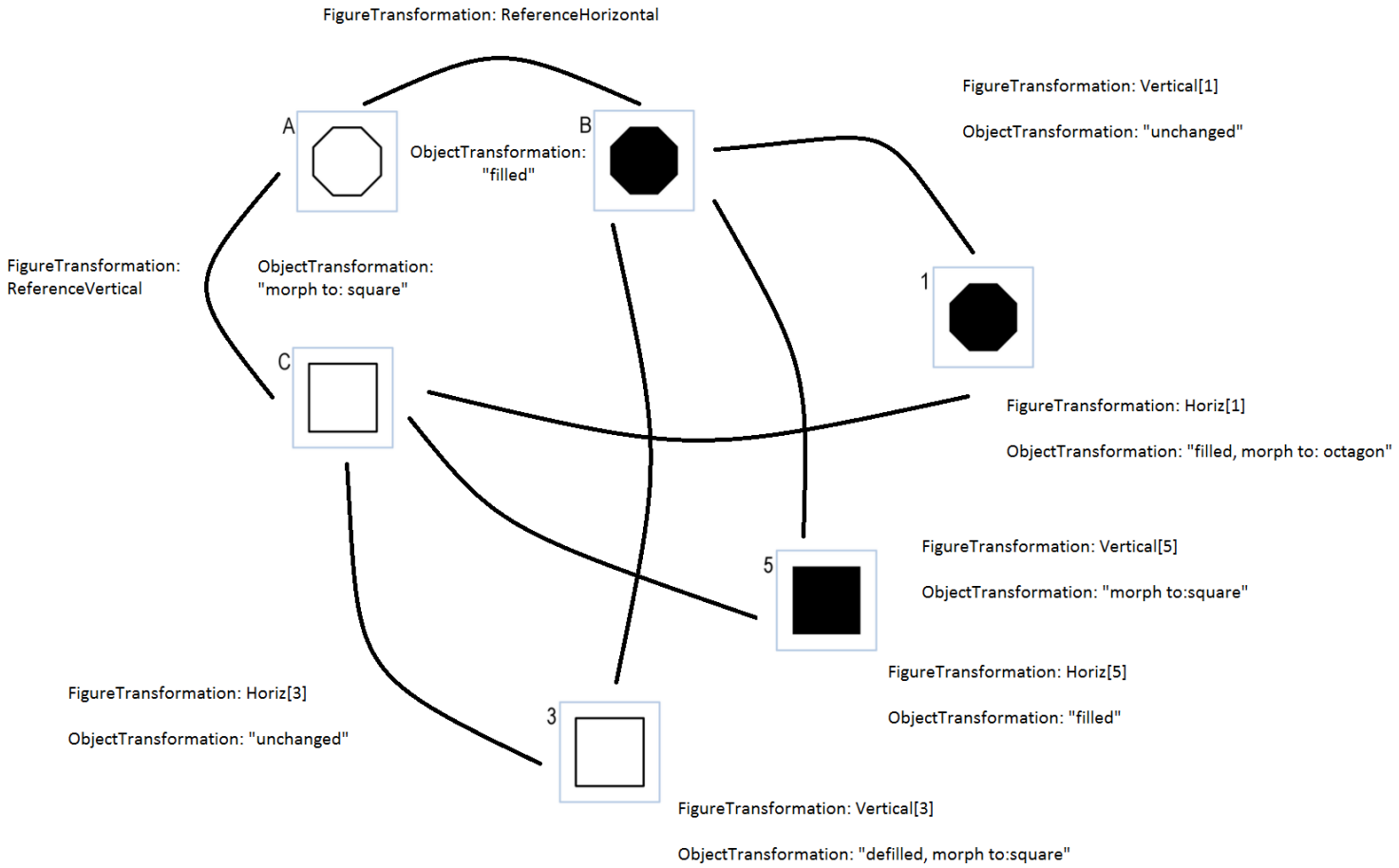

Figure 1. Visual representation of a sample semantic network (Basic B-09)

4.4.2: Problem Reasoning and Solution Selection

With a complete semantic network of object and figure transformations available, the strategy applies a simplified Generate and Test methodology to arrive at a best solution. In the case of RPM, the possible solutions to test have been generated already. The strategy follows the specific steps below to apply reasoning and solve the problem:

5. Each potential solution is compared the horizontal and vertical reference FigureTransformation objects, and a confidence level is calculated.
    a. When FigureTransformation objects are compared they:
        i. Determine which ObjectTransformations correspond to each other
        ii. Compare the ObjectTransformation objects to each other
6. The answer with the highest overall confidence level is returned.

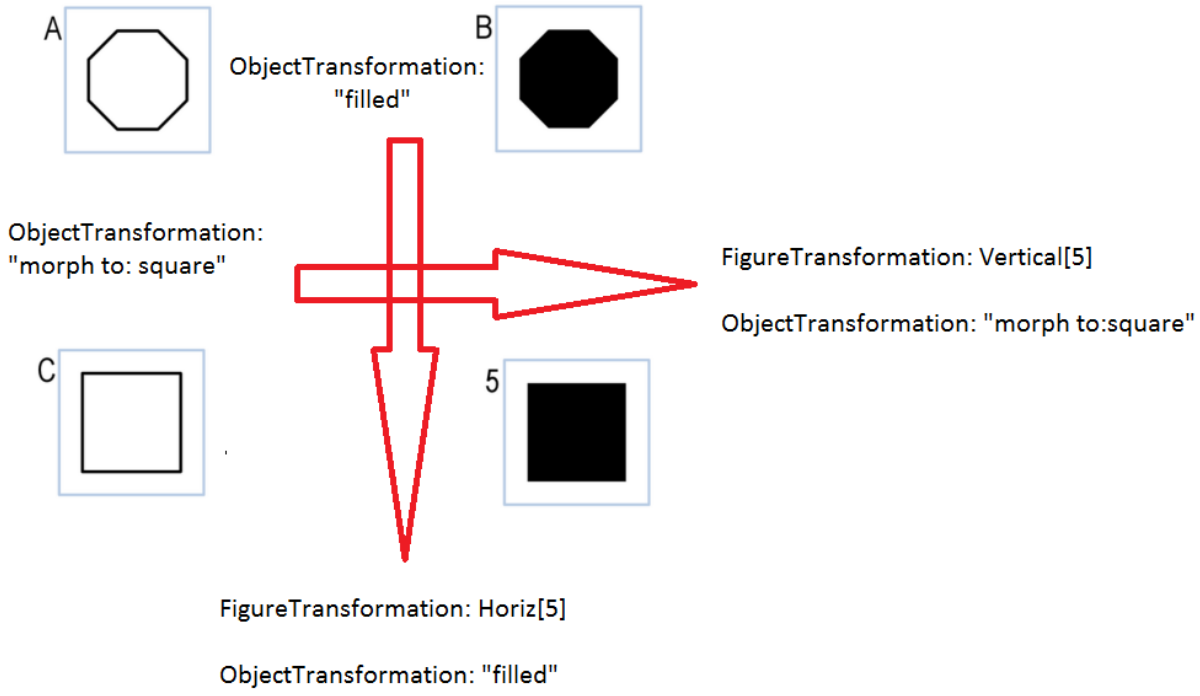

Figure 2. Comparison of ObjectTransformation values @ 100% confidence

4.3: Strategy Performance

Given the complexity of the problems this strategy is designed to solve, it performs well with respect to accuracy, generality, execution time, and algorithm complexity.

For problem set Basic B, the agent correctly identifies the answer to all 12 problems, with the correct answer having significantly higher confidence than the next nearest answer. The lone exception is problem B-04. This is due to the fact that one solution satisfies rotations, while the other satisfies reflections. In order to get the agent to properly identify the correct answer, a slight preference for reflection versus rotations was built into the ObjectTransformation comparison code.

Due to challenges in generating object correspondence, this strategy performs worst-case at $O(n^2)$ complexity, where n is the total number of objects in the problem. This strategy will perform more poorly with respect to execution time when large numbers of objects are present.

4.4: Strategy Weaknesses

4.4.1: Problem Representation

The primary drawback to this strategy is that it requires preprocessing of the RPM problem objects into frames of verbal data. This is not currently an issue, but later implementations of this agent will need to

process visual information. This strategy could be equipped with a visual parser that creates the verbal frames from the images in order to address visual only problems.

4.3.2: Edge Cases

The single largest risk I took in the design of this agent is permitting the agent to be "rough around the edges". More specifically, there are several edge cases in the agent's conditional logic that are not accounted for. For example, if two potential ObjectCorrespondence objects are found at the same confidence, there is no tie breaking, the first correspondence is used.

I realized while developing the agent that creating an exhaustively logically complete agent would be infeasible, if not, impossible. Human cognition is likewise scruffy. Humans don't always search for the perfect solution, in fact our problem solving favors finding the best solution possible with the smallest amount of effort possible.

In the end the risk paid off. Rather than spending time developing around edge cases, I spent the effort programming the agent to recognize a wider variety of transformations, fine tuning object correspondence, etc. The agent can correctly solve all Basic B problems in a relatively short execution time.

4.3.3: Strategy Mistakes

If verbal data was available for the challenge problems, the strategy would in its current form mistakenly identify an object with One to Many correspondence. Currently the correspondence engine only supports one to one relationships, and would mistakenly register object deletions or creations.

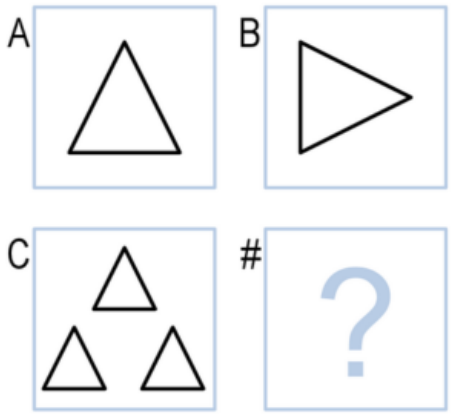

Figure 3. One to Many object correspondence

5: Closing Comments

5.1: Strategy Improvements

Given infinite time and resources, additional strategies would improve the agent's abilities to answer a wider variety of problems.

A visual 2x2 strategy that performs analogical reasoning on images without converting the problem knowledge into frames would help the agent solve RPM when nonsense shapes or various odd fills are utilized.

With additional strategies at the agent's disposal, a feedback learning mechanism would help this agent learn to rate strategies that solve the same problem. For example, if two different answers are returned at the same confidence level by different strategies, the agent could incorporate the checkAnswer() function into its overall testing methodology to determine which strategy arrives at the correct answer in a tie. This would enable the agent to mimic the human ability to learn from mistakes.

Additionally, to improve performance, especially under complex conditions, multithreading the agent to allow for strategies to be solved in parallel would ease the computational burden associated with the Strategy design pattern.

5.2: The Cognitive Connection

The single greatest insight into human cognition I have taken from this project is how implicit our problem solving skills are. When developing the agent to solve a new problem, I was forced to break down my mental process into small actionable and computable steps in order to develop an algorithm that could mimic it. I found this difficult, as these small actionable steps are so implicit they could be described as automatic.

While this agent finds solutions from a pool of available options like a human does, I would argue it doesn't actually solve the problem exactly like a human would. For example, when I solve these problems I rapidly detect patterns, eliminate certain responses without solution analysis, and could also generate the solution myself without a set of options to choose from.

While the agent could be updated to accomplish this as well, it would still suffer from the fact that its execution must reach a termination point. I took for granted that the human mind can solve simple problems like these using multiple approaches that can be initiated and dropped when it becomes clear that the solution will not work, an implicit action that is very difficult to incorporate into a computational system.