Solving Raven's Progressive Matrices With Semantic Networks

Magahet Mendiola

January 18th, 2015

Contents

Introduction

We will explore the design of an AI agent capable of solving Raven's Progressive Matrices. This agent will use Semantic Networks for knowledge representation.

Knowledge Representation

Semantic networks are a form of knowledge representation that reduces the complexity of solving a given problem. It can be thought of as a dimensionality reduction transformation; the output of which is more informative, tractable, and efficient to solve computationally. Semantic networks accomplish this by describing the problem in a way that is limited to the essential information required to solve the problem, without losing any information required to easily, and fully, understand the salient details.

Design Considerations

As an overall design goal, our semantic network representation should make it possible to describe the relationship within and between each frame of Raven's Progressive Matrices. This same descriptive model can then be applied to the target frame and each possible solution. The solution with the most similar description to that of the reference frames can be considered the best possible solution. We will review each concept in this process in the following sections.

Labeling

As a specific form of knowledge representation, semantic networks are defined as being made up of nodes, links, and labels. Constructing a semantic network to describe a problem involves assigning application specific meaning to that vocabulary of symbols. In the case of Raven's Progressive Matrices, one mapping could be made by assigning each object in a figure to a node; relationships between objects, as well as possible object transformations between figures, can be denoted with links and distinguished with labels.

Visual analogy problems, such as Ravens' Progressive Matrices, are well represented with a specific type of semantic network: a geometric analogy network. This defines nodes as shapes represented in a figure, links which identify relationship between the shapes, and other links showing the transformations from one frame to another. Relationship links are defined as one of the following:

- inside
- above
- to the left of

Transformation links are defined as a combination of the following:

- addition
- deletion
- expansion
- contraction
- rotation
- reflection

Using this vocabulary, we are able to frame Raven's problems in a way that quantifies the changes between frames. These changes can then be compared with those required from the target frame to each of the answer choices.

Similarity Metric

In the event that source and target frames have multiple possible transforms, it is important to establish a method for choosing one. This can be accomplished by setting weights for each transition link in a given semantic network, and preferring the semantic network with the best (smallest or largest) sum of transformation weights. Valid semantic networks can be generated by iterating though each transformation and object assignment, starting from the one with the best weight. Each valid semantic network can be applied to the target frame; the resulting figure then being matched against each answer choice. Since transformations are generated in rank order, the first one that generates a match from the target figure to an answer choice can be considered the best solution.

Correspondence

Each identified object must correspond to either an object in the target frame or have a "deletion" transition link. In order to find the optimal solution (the one with the best weighted match), we must try each combination of object assignments. The time complexity of traversing this search space can be reduced by adding heuristics for declaring a solution early. One such rule could be to stop when a generated transform perfectly describes the relationship between the target frame and one of the answer choices. This works if transforms are evaluated in rank order of their combined transformation weights. We could even restrict the agent by time and return the current best weighted match, assuming an unordered evaluation.

Matching

After establishing a method to generate semantic networks from sample frames, we then need a way to compare them to the target frame and the provided answers. We could generate the full set of transforms from the target frame to each answer beforehand and match each against the sample frame transform. This is best suited to problems where we believe the number of transforms between the target frame and the answers is low and the number of transforms between the sample frames is high. Alternatively, we could take the generated transform from the sample frames and apply it to the target frame; we would then match the resulting figure to each of the answer choices. Our choice of evaluating matches could be guided by prior knowledge or an initial evaluation of the problem features to make an educated guess as to which method would find a solution with the least number of network generations and evaluations.

Performance Evaluation

This method of searching the problem space is best suited to Raven's problems that require the least complex transforms; the search progresses in testing transformations in order of complexity. A particularly evil test writer could provide only valid solutions that require the worst weighted set of transformations. This would require our agent to evaluate every correspondence combination, as well as every transformation combination, before finding a solution. The agent would be rendered useless if restricted in running time or computational resources.

Improvements to this version of the agent could be to add a form of randomized search algorithm to traverse the problem space. This would be feasible if a given transform can be evaluated for how closely it came to matching transforms between the target frame and the answer choices. We could then use hill climbing to generate semantic networks that were similar to the previous one but moving toward better (both in weight and matching given solutions) transformations. A genetic algorithm could even be used to combine features of the best transforms in the search for a perfect, or perhaps just the closest, match. This robust search could still fail to find a perfect match, but could make an intelligent guess based on the current closest matching transformation.