Symantic Network Agent Implementation

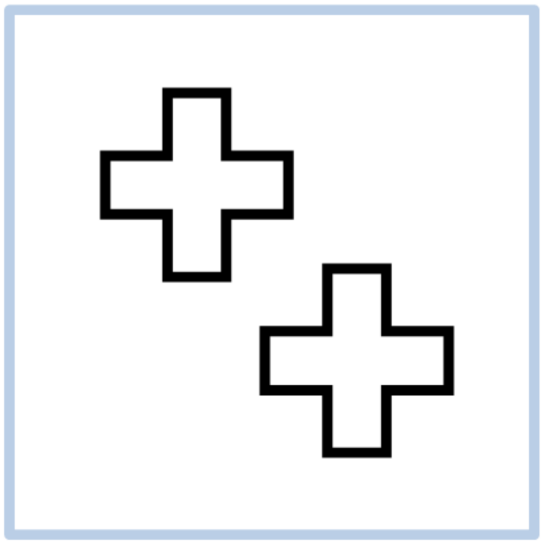
Assignment 1
Steve Saarinen
saarinen@gatech.edu
8/29/14

Problem

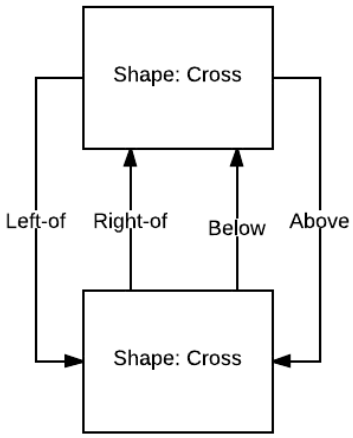
With a goal of goal of solving Ravens Intelligence tests in a variety of modes, implement a software agent utilizing Semantic Network knowledge representation and processing. A Semantic Knowledge network illustrates knowledge of current state and transitive state for objects using a graph consisting of nodes and edges with two classes of edges, state edges representing knowledge at a set time, and transitional edges, representing state changes for a specific node.

Modeling

Modeling state information and relationships is relatively straightforward. Take for example the following frame:



Nodes and edges like the following can model this simple frame state:



State graphs like this can be developed for each frame in the Raven series. Modeling the transitive relationships, however, are not as straight forward.

In order to model the transitive stages, we need to create a differential of the state of an object in the frames A and B, and link based on attribute name, using the difference in values as the label for string attributes and numerical differentials for properties such as angles. Boolean properties can be any value as their presence as a transitive property indicates a switch. By using a composition as a value, we eliminate having to create a separate lexicon for describing change, using terms such as “enlarged”, or “moved”.

Defining a Lexicon

Several Ravens problems involve transitions between defined shapes. As a computer does not have an inherent understanding of the physical attributes making up a shape, developing a lexicon for describing terms. For example, consider a square. A defined object may look like this:

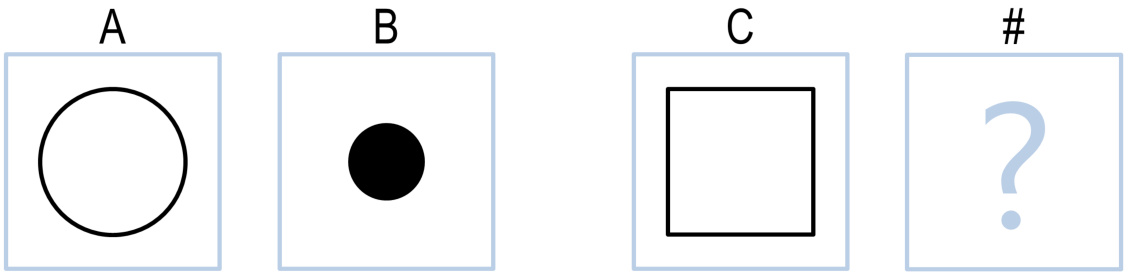
Shape: square
    sides: 4
    angles: [90,90,90,90]

Comparing it to a diamond:

Shape: diamond
    sides: 4
    angles [60, 120, 60, 120]

Processing

Once we have our data modeled and the lexicon defined, processing the Ravens series should be as simple as reducing the problem set to the delta between A and B, and applying those transitions to C. In an ideal world, we should be able to generate a model for D equating to a single answer in the set. Consider the following as an example:



The first step is modeling all frames. Once we have modeled A and B, we can calculate the difference. The resulting transitive edges look like the following:

Size: large => small
Fill: false => true

Applying these transitive edges to frame C, we can generate the ideal answer:

Frame C		Frame D
Shape: square	==	Shape: square
Fill: false	=>	Fill: true
Size: large	=>	Size: small

Comparing these to the possible answers, we find a match. This process should accommodate single, exact answers, but may fall short on either answers that are not exact, or multiple correct answers. Given small changes to our algorithm, we can make a best guess to address these shortcomings.

In the case where we do not have a choice for an exact right answer, we can run the same process we used to determine the transitive edges between our A and B frames, substituting our generated exact match against all possible answers. The best guess will have the least amount of differential edges. We are again foiled if we have more than a single remaining answer after this process. Fortunately the process for choosing between multiple close answers, and multiple exact answers are the same.

In the case where we need to make a guess given multiple exact answers, or multiple non-exact match answers, we need determine the differential in attributes between our candidate choices, and apply a scoring system to determine which answer to be "best" in a completely subjective case. By revisiting our lexicon generation step, we can build a map of attributes and user defined scores for which transitions are determined to be the least disruptive from a subjective human standpoint. Applying these scores to the differential from our candidates should always result in a single answer.

Learning

There are multiple opportunities for machine learning for the Semantic Network Agent. The most obvious application is modifying or generating the scoring matrix used to break ties between exact matches, or multiple imperfect matches. Given a non-initialized matrix, the Agent can initially make a guess at the

correct answer, check the answer, and modify the attribute scoring based on the result. Given a human initialized matrix, the Agent can adjust scores based the same criteria, the benefit to this situation being that the Agent's learning curve is not as sharp and better results will be found early in the process.