Generate and Test Agent Implementation

Assignment 2

Steve Saarinen
saarinen@gatech.edu

9/07/14

Problem

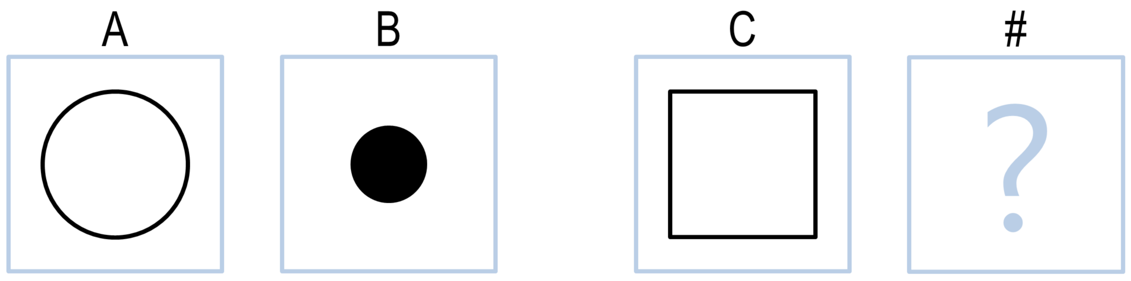
With a goal of goal of solving Ravens Intelligence tests in a variety of modes, implement a software agent utilizing the "Generate and Test" processing strategy. The Generate and Test strategy involves utilizing two disparate processing engines, a Generator, responsible for applying available transforms to a beginning state, and a Tester, responsible for determining whether the generated state matches the goal.

Modeling

Generate and Test concerns itself with processing and does not specify a data-modeling paradigm. As long as the generator and tester are working from, and can understand the same models, the processing strategy can be accomplished. Utilizing a semantic network for modeling may simplify processing steps as the transform edges are clearly defined.

Generation

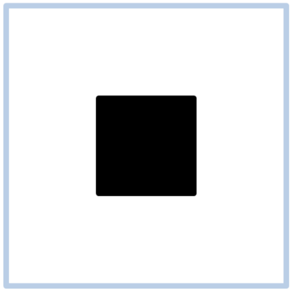
The Generator in our Ravens Agent has the opportunity to intelligently generate a set of possible solutions. Instead of generating random transform from an initial state and comparing it to the end state, we can deduce a set of transforms to apply given our initial frames. Take the following matrix for example:



The generator can deduce the transforms required to satisfy the difference between frames A and B. In this case, two transforms are immediately apparent:

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

Given these transforms, the generator can apply the same transforms to C to generate an ideal solution. The following figure illustrates the ideal frame for D:



With a target frame generated, processing transfers to the Tester to determine the best solution.

Testing

Now that our generator has provided us with a solution frame based upon the transformations detected between A and B, the tester needs to determine whether the generated frame matches any of the provided solutions.

By doing a straight attribute comparison between our generated frame and the solution set, 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 make our Tester more intelligent, 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, the Tester needs to determine the differential between our generated frame and our answer set. The best guess will have the least amount of differential transforms. 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, the Tester needs to 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 giving the Tester a map of attribute types and scores, the Tester can apply a scoring algorithm to our reduced answer set. Due to the fact that our answer set must be different from each other, the result should be 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.