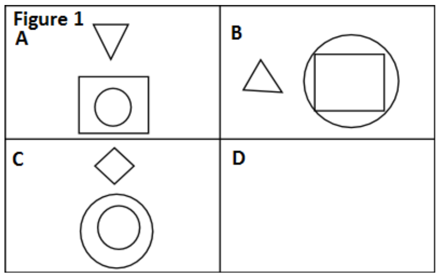
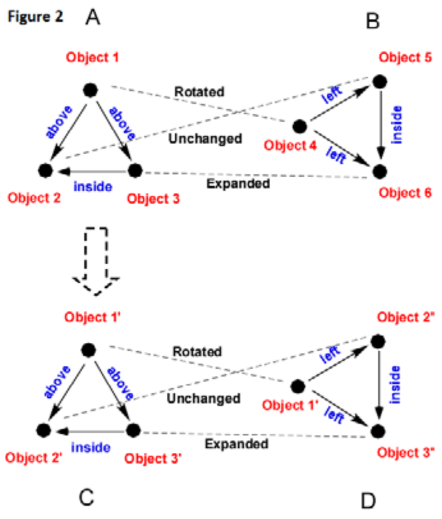
How to design an agent to address Raven's Progressive Matrices (RPM)?

Raven's Progressive Matrices (RPM) is a widely used nonverbal intelligence test. Considering that a child at age 5 can easily figure out the right answer, it should not be a challenge for computers since they have more powerful calculation abilities. That is what I thought before having any knowledge of artificial intelligence, and I realized its difficulties as soon as I started to design an agent to address RPM problems, and the computers' super calculation abilities cannot help, and on the contrary, maybe misleading for solving these problems. Computers are good at handling quantitative tasks. They can do millions calculations in a second and deliver accurate results. However, RPM is examine the ability which is kind of vague, and not quantitative. It asks the examinee to identify "difference", which itself is imprecise, between patterns of shapes, then apply the difference to figure out the missing element. That kind of ability belongs to human intelligence, which allows human being to "distill" valuable information from complicated environment, and discover the "internal links". So the challenges for AI agents focus on 1) how to extract valuable information (and dispose non-relevant information), and 2) how to compute a non-computable problem.

Semantic network provides a solution to address RPM, changing the original complicated non-computable problems into simplified quantitative and thus computable problems. There are three steps involved: first, isolate objects from complicated patterns of shapes, second, identify the relations between objects, e.g. one is above the other, third, identify the relations between states, and the relations (operations) are described in a quantitative way. E.g. 5 points for unchanged operation, and 4 points for reflected operation. The all three steps can be considered as extraction processes. Step 1 extracts complicated patterns of shapes to a list of objects. Step 2 simplified the relationships between objects. For a two dimensional object, each point on the object can be accurately described as (x, y), and computers can easily identify objects' positions' differences by calculate their different (x, y). However, such accuracy seen un-necessary. Human examinee obviously can identify the relationship between objects without such accuracy. So here, the relations between objects are simplified as one above, below, left, right and inside the other. Step 3 simplified the "difference" between states to a limit number of operations, e.g. unchanged, reflected, rotated, scaled, et al. It is worth to point out that these steps extracted information from complicated environment, and simplified the problem, but at the same time lose information that maybe necessary for solving problem. A highly intelligent AI agents should revisit these extraction steps and modify some of them when original search failures. The step 3 is especially important, which changes non-computable and vague term "difference" into quantitative and thus computable numbers. The setting of points of each operation is adapted based on human being's experience. For example, two identical squares are more likely to be considered by human being as after unchanged operation, vs reflected operation, vs rotated 90 degree operation … even though these operations deliver same results.

Although semantic network provides the general idea, there is still lots of detailed information need to fill in. Step 1, we need a program to transfer patterns of shapes into a list of objects list1 = [object_1, object_2, object_3]. The algorithm for the transformation is difficult, and I would not discuss the details here and assume such programs will be provided. By using these programs, we directly transform patterns of shapes (in Figure 1A) into objects (in Figure 2A). For example, the triangle in Figure 1A is described as object_1 in Figure 2A. The object class has variables such as: shape, size, center and angle. E.g. object_1.shape = 'triangle'. The object class also has a build-in method to describe their related positions, such as object_1.position(object_2) = ‘above’, based on objects’ size and center. Use the same procedure, Figure 1B will also be transformed into a list of objects list2 = [object_4, object_5, and object_6]. To compute the “difference” between Figure 2A and Figure 2B, we first need to pair up objects from these two lists (list1 and list2). This can be easily done if there are no shape changes, by sorting list2 via objects’ shapes. If the length of list2 is short than list1, and thus some object in list1 cannot identify its pair in list2, the related position in list2 should insert a ‘None’ value. Things get complicated when objects’ change shapes, thus a scored operation function has to be involved. The operation function takes two parameters (two objects), through computing objects’ variables, and return a number. For example, operation(object_2, object_5) should return 5, since we define the operation between two objects that have same shape, size and angle as an unchanged operation with 5 points, while operation(object_1, object_6) should return 3, as we define the operation between objects that have same shape and size, but different angles to be a rotated operation with 3 points. List2 should be sorted that the sum of paired operations between list1 and list2 reach its maximum. The difference between Figure 1A and 1B final represents by two things 1) total operation scores, 2) objects’ position changes. Figure 1C and all candidate solutions will be computed and compared the same way, and the candidate solution that has closest operation score, and same objects position relations will be the right answer for RPM.





I would also slightly touch on the second topic **Generate & Test**, which is a different algorithm used to solve RPM. The algorithm that I just described directly compare Figure 1C with all candidates, and identify the best answer rely on scores generated by the operation function, while the Generate & Test strategy will first generate an ideal solution, then compare the ideal solution with all candidates to find the match solution. The first extra step is to pair up objects in list1 and list3, and this pair up procedure is different from the pair up between list1 and list2. Objects’ shapes are more likely changed, thus the order of objects in list3 may heavily rely on objects position relations. In list1, object_1 is above object_2, 3, and object_3 is inside object_2, thus in list3, objects should follow the same order, that object_1’ is above object_2’ 3’ and object_3’ is inside object_2’, even though their shape, size and angle are totally different. Objects in list4, the ideal solution, are copies of objects in list3. Some changes in shape, size, angle and position are further applied on each object through comparing objects in list1 and list2. Finally, via comparing the ideal solution with candidates solutions to find the best solution. You may wonder what's the advantages of Generate & Test strategy, since the number of comparing is not reduced, and plus the time spent on generating ideal solutions, and why human beings like the Generate & Test strategy to solve RPM. A possible explanation is that after generate the ideal solution, the comparison between ideal solution and candidate solutions will be much easier, and may not require the scored operation function (e.g. by just comparing their shapes, or position relations).