Semantic nets for Ravens progressive matrices

Rohan D. Kekatpure

1. Problem description

Assignment 1 is intended to provide a high-level design plan as well as an implementation path for Project 1: Building an AI agent for solving 2×2 Raven's Progressive Matrices (RPMs) using **visual** repreentation. In addition to design, we'll also address potential implementation challenges (given the project constraints) and explore how knowledge representation using semantic networks (semnets) helps us address some of the identified problems. The formal Project 1 description can be found in the course material and does not need repetition. The problem description presented here will instead focus on the project sub-areas that need to be addressed successfully to solve the overall assignment.

Programming an AI agent to solve RPMs using visual representation presents *two* distinct challenges. First is the identification of basic shapes in (grayscale?) images. The second problem is choosing a semnet representation and constructing it from the shapes found in the pair of images. Once these problems are addressed, we can build a semnet representing the relationship between two images. Using the semnet, one can evaluate the candidate answers using either the 'generate-and-test' or 'describe-and-match' techniques.

2. Areas of difficulty

The two subproblems described above can be explicitly divided into following list:

2.1. Detection of shapes in images

This is essentially a computer vision problem and could be addressed by using a library like OpenCV. However, since the guidelines prohibit libraries, we need to re-create a part of the OpenCV shape detection API.

2.2. Detection of relationships between shapes in an image

The next step after detecting shapes in a given image is the problem of detecting relationships between shapes in the *same* image. For instance "Triangle in (A) is **inside** the square in (A)". In other words, this step is generating the verbal description of a single image. A potentially hard part here is separation of overlapped shapes as seen in Basic problems C and D. The visual $\rightarrow$ verbal transformation is, in my opinion, the hardest task in project 1.

2.3. Construction of a semantic network

Note that addressing §2.1–§2.2 will provide the lexical, structural and semantic components required for construction of a semnet representation of the problem. Given the verbal representation of the images, we next need to choose an appropriate data structure for a computer representation of a semnet. We could then use this representation to generate candidate images and test against the provided answers.

3. High-level design

The design can be divided into following components that will need to be built to implement the overall solution.

3.1. Shape detection library

Since the ultimate objective is to build an agent capable of *visual* reasoning, a shape-detection library is essential. Observing the problems, it appears that the first version of the library would need to detect a subset of basic shapes: circle, octagon, square, 5-pointed star, heart, diamond, and a cross. Additional needed functionality would be an ability to detect fills and overlapped shapes. The output of the library would be a dictionary containing the number and description of shapes. For instance:

```
shapes = detect_shapes(image_array)
shapes = {
   "count": 3
   "list": [{"type": "circle",  "filled": "false", "size": "large"},
            {"type": "square",  "filled": "false", "size": "medium"},
            {"type": "diamond",  "filled": "true", "size": "small"}]
    }
```

At this high level of design it is not necessary to dig deeper into the implementation details of the shape-detection library. However, note that the existence of OpenCV library implies that this task is feasible. We'll essentially rebuild the required parts of the OpenCV library in Python.

3.2. Semantic network representation

This component is where we combine the verbal descriptions generated above with domain rules (a.k.a **heuristics**) to build semantic representation of the problem. One possible way is to simply **sort** the shapes in an image using the number of sides, size, and fill condition and so on, as successive sorting keys. In this representation, the order in which shapes are presented is always `[circle, heart, octagon, star, square, diamond, rectangle, triangle]`. Further, similar shapes are ranked according to size, with filed shapes after clear shapes. Although the full set of heuristics is required at this point, it is important to note that it must be consistent.

The sorting allows representation of a semantic network as a similarity vector between images. For instance consider two images on which we have performed shape detection:

```
shapesA = detect_shapes(imageA)
shapesB = detect_shapes(imageB)

shapesA = {
   "count": 2
   "list": [{"type": "circle",  "filled": "false", "size": "large"},
            {"type": "octagon",  "filled": "true", "size": "small"}]
    }
shapesB = {
   "count": 2
   "list": [{"type": "circle",  "filled": "true", "size": "large"},
            {"type": "octagon",  "filled": "false", "size": "small"}]
    }
```

We then loop through keys of shapesA and shapesB to compare counts and the attributes of each shape and generate a similarity vector as follows:

```
simAB = []
simAB.append(shapesA.get("count") == shapesB.get("count"))

shape_attrs = ["type", "filled", "size"]
for sA, sB in zip(shapesA.get("list"), shapesB.get("list")):
    for attr in shape_attrs:
        simAB.append(sA.get(attr) == sB.get(attr))
```

For our example above, the similarity vector (after converting boolean to ints) would be `simAB = [1, 1, 0, 1, 1, 0, 1]`. Assume that our image C yields,

```
shapesC = detect_shapes(imageC)
shapesC = {
    "count": 2
    "list": [{"type": "square",  "filled": "false", "size": "large"},
             {"type": "diamond",  "filled": "false", "size": "small"}]
     }
```

With that we can compute the similarity between image C and each of the candidate images (`simCD`) and choose the image D that yields closest match between `simAB` and `simCD`. We can take a similar approach in the vertical direction and match `simAC` and `simBD`

3.3. *A simpler but less powerful alternative*

An alternative approach would be to **enumerate** all possible shapes and populate a fixed sized vector for each image. For instance the components of the master vector would be existence in given image of each of the enumerated object types:

```
master_vector= \
    [is_clear large circle_present,
    is_clear_large_heart_present,
    is_clear_large_octagon_present,
    ...
    is_clear_medium_star_present,
    is_clear_medium_circle_present,
    is_clear_medium_heart_present,
    is_clear_medium_octagon_present,
    is_clear_medium_star_present,
    ...
    is_filled_large_circle_present,
    is_filled_large_heart_present,
    ...
    ...
    ]
```

This list would be small; hundreds of elements at max. For images A nd B, we calculate a 'diff' of using the master vector of each image. Similarly we calculate the 'diff' of image C with each of the candidate images and choose the one such that

$$|\text{diff}(A,B)| \approx |\text{diff}(C,D)|$$
$$|\text{diff}(A,C)| \approx |\text{diff}(B,D)|$$

Note that this approach, while simple, ignores the inside/outside relations. It'd be easy to cheat this agent using choices that contains the same, but differently arranged images. We could improve this approach by adding shape location to the shape information.

4. How semantic networks are leveraged to solve the problem

The key realization, as mentioned in the lectures, is that semnets are merely knowledge representation construct. By themselves, they cannot be leveraged to solve any problem. They need to be coupled with a problem solving method.

We propose to represent the semantic relations between an image pair as a "similarity vector". The difficult part is the construction of this similarity vector given an image pair. Given the similarity vector between A and B, and an image C, we can assign probabilities to the candidate images. These probabilities will be proportional to the absolute value of the similarity vector between C and the candidate.

Note that other representations are possible and may perform better. Our objective here is to simply sketch a first-pass solution. The final solution will require changes and iterations to refine the above approach. For example, an alternative semnet representation might use a graph data structure to represent the nodes and labeled, directed edges. We might then construct graphs with image C and the answer candidates and define some notion of ‘graph similarity’. The approach is equivalent to the above vector representation.

5. Feasibility of proposed solution

The main feasibility concern for implementation is the generation of verbal description from the images. We can address this concern in two ways. First we restrict our solution to a subset of shapes: namely circle, octagon, star, heart, square, diamond and triangle. We cans also discretize size (small, medium large) and angle (0, 90, 180, 270). We then need to implement algorithms for extracting edge pixel co-ordinates and map those to a (shape, size and angle) tuple.

The shape extraction functionality already exists in the OpenCV library. Therefore the major component of Project 1 would be the image $\rightarrow$ verbal conversion.

6. What about 3×3 problems?

Most of the above discussion was focused on 2×2 matrices. One approach to a 3×3 problem is to divide it into a series of 2×2 problems. For example, consider a 3×3 problem represented as a series of images labeled from A ... I. The image detection, image $\rightarrow$ verbal description, and similarity computation are re-used from the 2×2 version.

There are now multiple heuristics for matching the candidate answers. In one approach, we compute similarity between vertically and horizontally adjacent image pairs and extend the similarity notion to three images as:

$$\text{sim}_{ABC} \equiv \frac{1}{2}\left[\text{sim}(A,B) + \text{sim}(B,C)\right] \tag{1}$$

and choose the final image I such that

$$\begin{aligned} \text{sim}_{ABC} + \text{sim}_{DEF} - 2\text{sim}_{GHI} &\approx 0 \\ \text{sim}_{ADG} + \text{sim}_{BEH} - 2\text{sim}_{CFI} &\approx 0 \end{aligned}$$

Another possible approach is to ignore the leftmost and topmost images and consider the images E, F, H and I as a 2×2 problem. This would be faster but more error-prone.

Without performing actual experiments, this is about the maximum detail we can go into without the risking too much speculation. It is understood that the final approach will be optimized using iterations and may be significantly different from the one presented here.