RPM Milestone 2 Journal

Xinghao Chen
xchen785@gatech.edu

***Abstract*—**

1 THE SCHEMATIC OF MY AGENT

My agent is built with 3 layers: the computer vision (CV) module, the relationship extractor, and the analyzer of relationships. Based on robust CV functions, the agent can obtain geometric descriptors of each connected component, and identify relationships between components including unchanged shapes, symmetry and filling. With relationships extracted, the analyzer of relationships then score the similarities between relationships and selects the answer with highest relationship similarity.

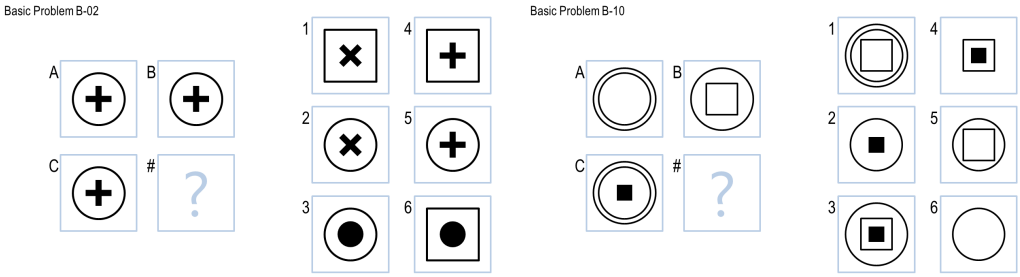
1.1 The computer vision infrastructure

1.1.1 *Connected components*

A connected region is a single shape where you can rout from one point inside the shape and reach any other point in the shape without leaving the shape. With this concept, we can easily extract components from basic B problems using the function "cv2.connectedComponents". Figure 1 shows some examples where the idea of connected components works very well. However, we should notice that connected componets do not work well in all the RPM problems. See Figure 2 for examples. Fortunately, in the basic B set, we do not encounter such problems.

1.1.2 *Erosion, dilation and the open operation*

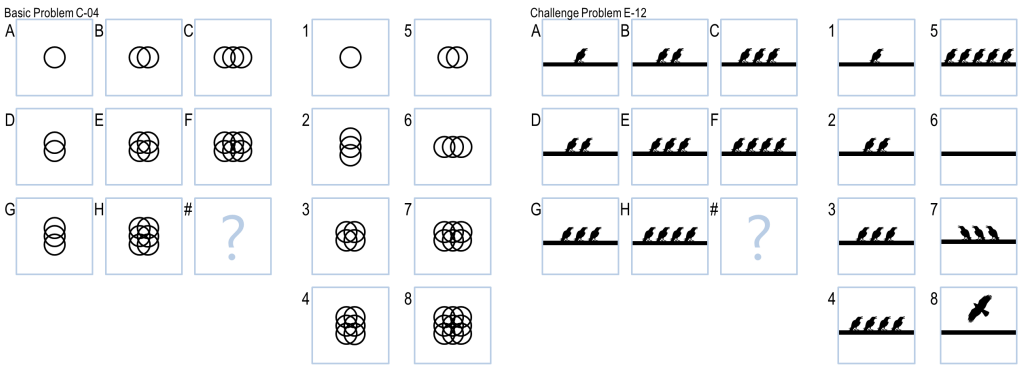
Erosion is to remove a small thickness of pixels on the contour of a connected region, while dilation is to add an extra small thickness of pixels on the contour. If we, by using the open operation, first remove and then add pixels to the edge of a connected region, the region is usually preserved the same as before. However, if the original component is thin enough (for example, a straght line), it would disappear after the erosion.
The open operation can be typically executed with "cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel=cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3)))". Usually the operation is taken on the difference between two images of



(a) Connected components: circles, squares and crosses

(b) More connected components: circles and filled or unfilled squares

Figure 1—Samples of problems where the concept of connected components generalizes well



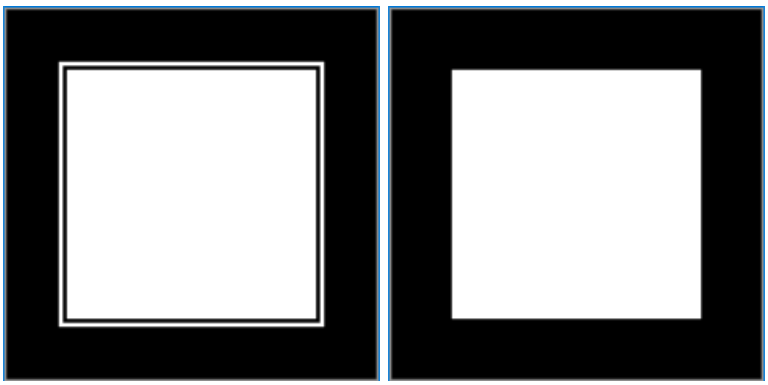
(a) A case where conceptively discrete circles are morphologically connected

(b) A case where we have to identify semantic components from a single morphologically connected region

Figure 2—Samples of problems that connected components cannot solve

single connected components.

By controlling the thickness of erosion and dilation, the open operation is an efficient method to neglect minor difference between two shapes and to declare the shapes as "morphologically the same". For example, in basic problem B-09, if we compute the absolute difference between figure C (an unfilled square) and figure 5 (a filled square), we get the result in Figure 3(a). However, the open operation can eliminate the thin, unfilled square outside the filled square, and yield only the filled square shown in Figure 3(b).



(a) The absolute difference

(b) The morphologic difference generated by the open operation

Figure 3 — An example of using the open operation at a thickness of 3 pixels

1.1.3 *Filling detection*

The detection of filling is achieved with the following steps, where open operations are always taken in checking morphological differences, at a thickness 3:

- Input two figures, each including only 1 connected component.
- Check whether the two figures has any intersection of pixels. If there is no intersection, we just declare that there is no filling relationship.
- Compute the image of morphological difference d between two figures.
- Extract each single connected component in d as a single image d_i. Generate images of the contours of each d_i at a thickness of 3.
- Check whether any contour of d_i is morphologically the same as a contour in the two input images. If a contour in an input image is morphologically the

same as a contour in the morphological difference image, then the contour in the input has been filled.

The algorithm above can only detect fully filled regions, and cannot deal with patterns of filling (such as intensive parallel lines). Note that an unfilled shape has the inner contour and the outer contour. Using the algorithm above, we always judge whether the inner contour is filled.

1.1.4 *Symmetry detection, and detection of unchanged shapes*

These functions are achieved by simply computing the morphological differences between images. We can flip the image using "cv2.flip(fig2, flip_direction)".

1.2 The relationship extractor

For basic B problems, I divide each image into single connected components, and create a descriptor for each component. We can establish all the detected relationships between each pair of components. Some components may have no evident relationship with components in another figure. In this case we just declare that the component has "disappeared".

1.3 The analyzer of relationships

The task of this analyzer is to determine the similarity between two sets of relationships. In other words, "How similiar is A is to B as C is to D?". A score is counted for each element of similarity between two sets of relationships.The choice of the highest score is returned as the final answer.
In my analyzer for basic B problems, the score is only given if **the same type of relationships happens at equal counts** between A to B and C to D. For example, if 2 components disappered from A to B, but 3 components disappaered from C to D, my analyzer would give no point to these relationships of disapperance.

1.3.1 *Analyzer of disapperance*

The analyzer of disapperance gives 1 point of score if the disappeared components in figures A and C are morphologically **the same**.

1.3.2 *Analyzer of symmetry*

Sometimes there may be multiple types of symmetry (up-down, left-right, up-down-and-left-right) that stands for two single connected comonents. The analyzer of symmetry gives 3 points of score if the symmetry type in two sets of

relationships **can be** the same.

1.3.3 *Analyzer of unchanged shapes*

The analyzer gives 1 point for each pair of unchanged shapes.

1.3.4 *Analyzer of filling*

The analyzer gives 3 point for each pair of filled shapes.

2 PERFORMANCE

My agent roared rapidly thoough all the 12 basic B problems and 12 test B problems, making all of them correct. However, it almost does not generalize to other sets of problems at all. This is because the agent has no further knowledge (such as rotation) for the subsequent problems. Besides, the idea of connected components does not fit many problems in other sets.

3 PLANS FOR FUTURE IMPROVEMENTS

3.1 Generalization to 3×3 problems

The CV infrastructure may remain the same as what has been established in 2×2 problems, but the upper layers involving relationship extraction and analysis must be modified to adapt to more complex relationships.

3.1.1 *More possible relationships*

In 3×3 matrices, we may have to find relationships "**among**" 3 figures, instead of 2 pairs of figures. That is, there may be operators such as XOR that takes 2 images as the input, and gives the third image as the output. More knowledge about relationships should be injected into the agent to tackle such problems.

3.1.2 *A revised analyzer of relationships*

The difference between 2×2 and 3×3 requires the analyzer of relationships to be re-written. Sometimes the relationships between 3 figures are 1 by 1 by 1, which is similar to those in 2×2 problems. But in other cases, as have been mentioned in 3.1.1, complex relationships "**among**" 3 figures should be taken into consideration.

3.2 Possible improvements for component extraction

Connected areas do not serve many complex problems very well. We may build a knowledge base of commonly-seen shapes, and detect them in the problem using the templates in the knowledge base.

3.2.1 *Feature matching and object detection using Scale-Invariant Feature Transform (SIFT) and Fast Library for Approximate Nearest Neighbors (FLANN)*

An SIFT operator, which finds rotation-invariant and scale-invariant feature points, can be accessed with "cv2.SIFT_create()". We may match features from the knowledge base to those in the shapes appaering in the problem, using simple nearest-neighbor algorithms provided by "cv2.FlannBasedMatcher" to detect shapes.
Note that this method may require abundant features inside the shape itself. Simple features in a mere filled square may not be accurately detected.

3.2.2 *Unsupervised object detection and aggregation*

Referring to Figure 2(b), humans have the ability of extracting and aggregating/clustering similar shapes from the figure and counting them to find relationships between figures. I would recognize this ability as a key to many challenging problems. However, a machine without deep-learning networks may have difficulty in doing so. I have not come up with any plausible idea to do so, but an unsupervised, or pre-trained semantic segmantation function may be preferred in the future.

4 HOPES FOR FEEDBACK

4.1 How to segment components

Taking any connected region as a single comonent cannot survive the future. Therefore, I am looking forward to ideas to segment shapes semantically.

4.2 How to detect and describe different patterns of fillings

Some fillings in challenge problems has patterns like intensive parallel lines, instead of just fully fill a contour with black pixels. How to detect the fillings and describe them?