KBAI
Fall 2015

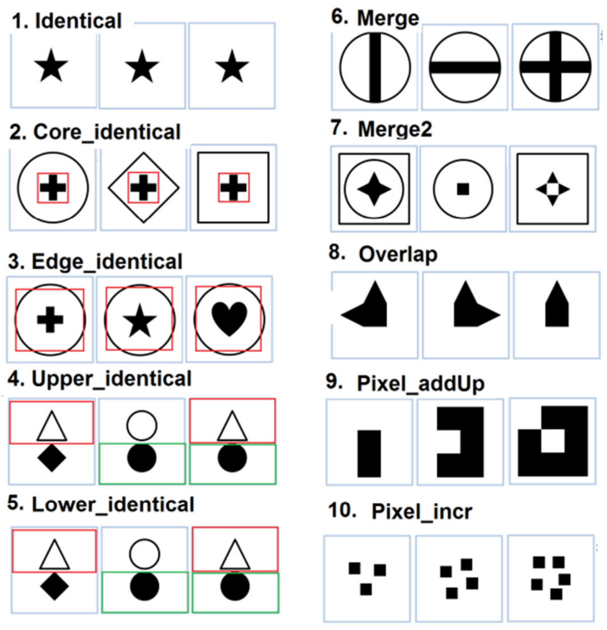
Project 3 Reflection

How does your agent reason over the problems it receives? What is its overall problem-solving process? Did you take any risks in the design of your agent, and did those risks pay off?

I used verbal representation in project 1 and project 2, and since project 3 is based on visual representation, I tried a different strategy. I found most problems in Basic problem D and Basic problem E can be resolved by using ten help functions (Figure 1) through four different comparisons (Figure 2).

The ten help functions are:

Figure 1: Basic Operations



1. *Identical* (examine whether images are identical)
2. *Core_identical* (examine whether images in core areas are identical)
3. *Edge_identical* (examine whether images in edge areas are identical)
4. *Upper_identical* (examine whether images at upper sides are identical)
5. *Lower_identical* (examine whether images at lower sides are identical)
6. *Merge* (examine whether one image is generated through merge the other two)
7. *Merge2* (a slight different operation compare with merge, that cancels overlap parts)
8. *Overlap* (examine whether one image is the overlap of the other two)
9. *Pixel_addUp* (combined pixels of the first and second images equal to the third image)
10. *Pixel_incr* (pixels difference between image1 and image2, equals pixels difference between image2 and image3)

I first transformed each image into a 2D array. The help functions are simply using two *for* loops to compare the two (or three) 2D arrays. Here is the *Identical* function, which returns a Boolean value after comparison between two 2D arrays (pixels):

```
def Identical(self, pixel_1, pixel_2):
    difs = 0  # number of different pixels

    for i in range(184 ):
        for j in range(184):
            if pixel_1[i, j][0] != pixel_2[i, j][0]:
                difs += 1

    if difs > 500:   # the threshold for true or false can be adjusted.
        return False
    else:
        return True
```

The four different comparisons are shown in Figure 2. For example, the Basic problem D-1, my agent is using *Identical* help function to examine whether A = B = C, C = D = E, and searching answers fitting G = H = answer.

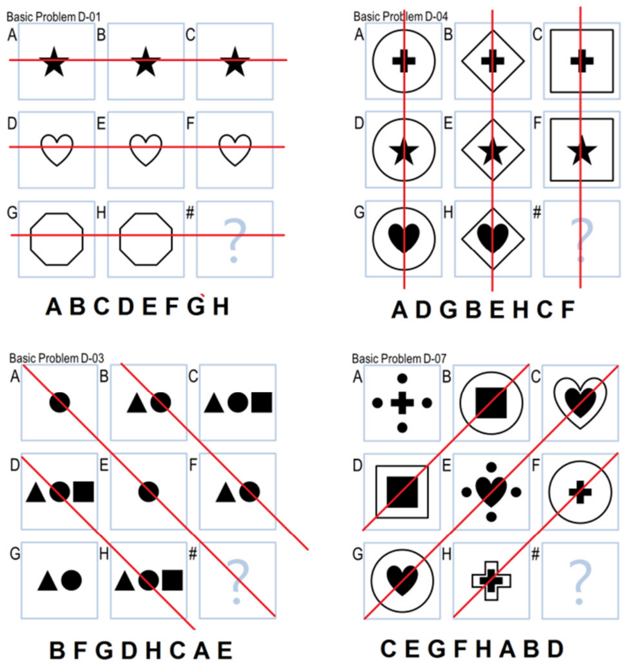
The general work flow is like this:

```
comparisons =  [ [A, B, C, D, E, F, G, H],
                 [A, D, G, B, E, H, C, F],
                 [B, F, G, D, H, C, A, E],
                 [C, E, G, F, H, A, B, D] ]

answers = [1, 2, 3, 4, 5, 6, 7, 8]

Identical operation
for c in comparisons:
    check whether c1 == c2, c2== c3, c4 == c5, c5 == c6
    if that's true, find all answers that ax == c7 and ax == c8
    update answers list
    if len(answers) == 1, return the answer
……
repeat same procedure for the other nine operations
……
Finally, if there are still multiple answers in answers list, return '-1'
```

Figure 2. Major Comparisons



After each comparison, answers that do not fit will be removed from the answers list, which leaves fewer comparisons at late stages. The advantage of this strategy is that it can greatly reduce comparisons during the whole run, and my agent usually does not need to finish all comparisons before it delivers correct answer (for the Basic problem D-1, only *Identical* help function is called and the other nine help functions are not used before delivering the right answer). The risk of this strategy is that if the correct answer is removed at earlier stages by mistake, then the search will fail after the agent finishes all comparisons after ten different operations (there is no make up for earlier mistakes). I do not want to exclude correct answer at earlier stages.

How does your agent actually select an answer to a given problem? What metrics, if any, does it use to evaluate potential answers? Does it select only the exact correct answer, or does it rate answers on a more continuous scale?

I used ten help functions combined four different comparisons to screen the correct answers. By removing incorrect answers from the answers list, there are few answers left in the list. My agent will return the only answer that left in the answers list.

Let's use the Basic Problem D-04 as an example. The original answers list contains all eight answers (answers = [1, 2, 3, 4, 5, 6, 7, 8]). There was no update for the answers list after called *Identical* help function, since there were no identical images. The list was updated after calling *Core_identical* help function, since in core areas, A = B = C and D = E = F. my agent removed 2, 4, 5, 6, 7 from the answer list because they did not fit G = H = answer. Answers list was updated as [1, 3, 8]. Answers list was further updated after calling *Edge_identical* help function, since in edge areas, A = D = G and B = E = H. my agent removed 3, 8 from the answer list because they did not fit C = F = answer. Then, there is only one answer left in the list, and my agent returned the only answer as correct answer.

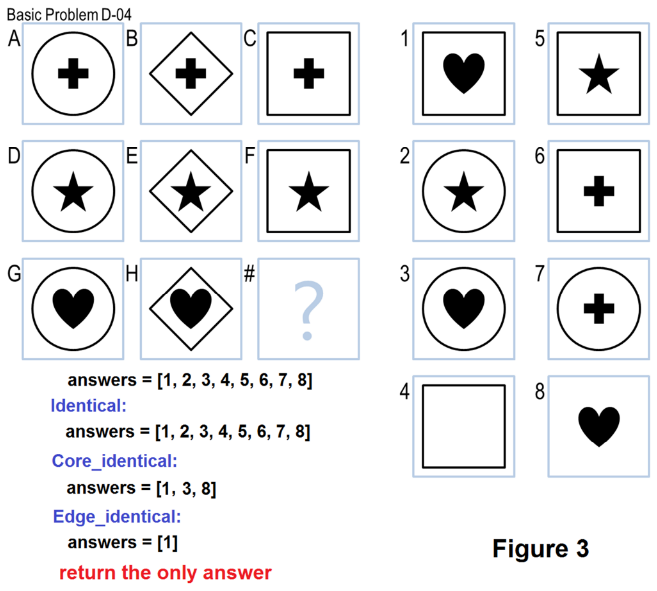


Figure 3

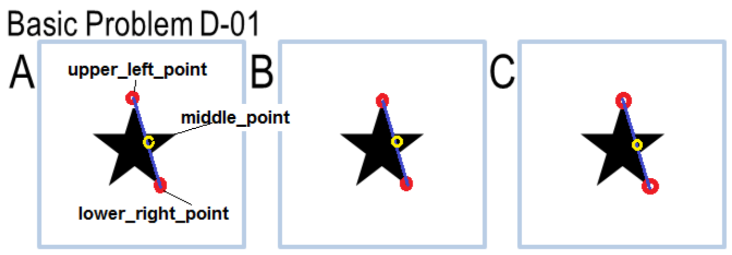
My strategy does not directly select the correct answer, instead it removes incorrect answers from answers list until only one matched answer left.

What mistakes does your agent make? Why does it make these mistakes? Could these mistakes be resolved within your agent's current approach, or are they fundamental problems with the way your agent approaches these problems?

As described, my ten help functions take two (or three) 2D arrays (pixels) and return a Boolean value (True or False) based on number of different pixels (difs). At the beginning, I set the cutoff line of difs at 50, which means two images are considered as identical only when they have less than 50 different pixels. Later, I found that setting was not practical. Most identical images (based on my visual examination) have more than 50 different pixels, and my agent provided lots of false negative results. I then increased the cutoff line to 100, 200, 500, or even 1000. The large cutoff number reduced false negative results, but provided more false positive results that return true value for non-identical images (based on my visual examination).

I only partially solved this problem through an alignment operation. For images A, B, C of Basic problem D-01, even though they look identical through visual examination, the number of different pixels (difs) is quite big due to their slight location changes. I created two extra help functions: *find_upperLeft_point*, and *find_lowerRight_point* (Figure 4). I used the middle point to align the images before comparisons. I found after adjustment, the number of different pixels (difs) is greatly reduced. The final cutoff line of difs is set as 500, and it works great for most problems.

Figure 4: image alignment



KBAI
Fall 2015

What improvements could you make to your agent given unlimited time and resources? How would you implement those improvements? Would those improvements improve your agent's accuracy, efficiency, generality, or something else?

My strategy is to remove answers that do not fit from the answers list after each round of comparison, thus my agent only need perform less and less comparisons. My agent usually does not have to finish all operations and comparisons before giving correct answer. The work flow is like:

```
        answers = [ "1", "2", "3", "4", "5", "6", "7", "8"]
   1:  Identical  → update answers = [“1”, “2”, “3”, “4”, “5”, “6”, “7”, “8”]
   2:  Core_identical → update answers = [“1”, “3”, “8”]
   3:  Edge_identical → update answers = [“1”]
   4: return the correct answer “1”
```

My agent is efficient by using this strategy. It finished all problems in Basic Problem D and Basic Problem E in about 2 minutes. ***Caveats:*** 1, if the correct answer is removed by mistake at any earlier stages, nothing can compensate that mistake. 2, the “correct” answer may be returned too early. If there is only one answer in the answer list, my agent will stop future comparisons and return the answer. What if a better answer is generated at later comparisons?

If I have unlimited time and resources, I would use a different strategy, and create a dictionary using answers as keys and their match times as the values. Finally, my agent picks the answer with highest score. The work flow is like:

```
          answers = [ "1", "2", "3", "4", "5", "6", "7", "8"]
          dict = { "1" : 0, "2" : 0, "3" : 0, "4" : 0, "5" : 0, "6" : 0, "7" : 0, "8" : 0}
   1: Identical operation
      update dict = { "1" : 0, "2" : 0, "3" : 0, "4" : 0, "5" : 0, "6" : 0, "7" : 0, "8" : 0}
   2: Core_identical operation
      update dict = { "1" : 1, "2" : 0, "3" : 1, "4" : 0, "5" : 0, "6" : 0, "7" : 0, "8" : 1}
   3: Edge_identical operation
      update dict = { "1" : 2, "2" : 0, "3" : 1, "4" : 1, "5" : 1, "6" : 1, "7" : 0, "8" : 1}
      ……………………..
   10: Pixel_incr operation
      update dict = { "1" : 2, "2" : 0, "3" : 1, "4" : 1, "5" : 1, "6" : 1, "7" : 0, "8" : 1}
  10: return the correct answer “1”
```

I can also sign different values for different operations, for important operations, I can give them higher scores. Through further refinement, my agent will be much more accurate compared with my current design but less efficient (more comparisons and longer running time).

How well does your agent perform across multiple metrics? Accuracy is important, but what about efficiency? What about generality? Are there other metrics or scenarios under which you think your agent's performance would improve or suffer?

My agent solved 12/12 problems in Basic Problem D and 12/12 problems in Basic Problem E in about 2 minutes. However, my agent's answer is purely based on the ten operations (ten help functions), thus if there are any new operations in the test problems, my agent would not give correct answers. My agent needs more training to expand the list of operations that it can recognize.

Which reasoning method did you choose? Are you relying on verbal representations or visual? If you're using visual input, is your agent processing it into verbal representations for subsequent reasoning, or is it reasoning over the images themselves?

I used visual representation in project 3. My original plan is to transform all images into verbal representation and solve them using the agent developed in project 1 and project 2. I found the transformation is very challenge. Thus I used the new strategy as described.

Finally, what does the design and performance of your agent tell us about human cognition? Does your agent solve these problems like a human does? How is it similar, and how is it different? Has your agent's performance given you any insights into the way people solve these problems?

My project 1, 2 used verbal representation, and project 3 uses only visual representation. What surprised me is that my strategies using verbal representation is closer to human cognition, and my strategy using visual representation is totally different compared with human cognition. Human eyes identify objects through their size, shape, or color. Due to kind of limitation, human cannot compare images at pixel level. While for computers, it is easy for them to compare the pixels than summarizing images by size, shape or color (pattern recognition). Such accuracy (pixel level) of computers turns to be a disadvantage in solving raven's progressive matrices problems, and I have to increase the cutoff line of the number of different pixels (difs) to bring more flexibility to match human's cognition.

When we develop AI agents, we hope these agents can obtain human's intelligence. However, there is something that computers can obviously do better. Should we keep these advantages of computers during agent design? For example, in movies, superman has super vision (probably at pixel level). So if he takes the raven's progressive matrices test, will the super vision bothers him and brings him a bad score, and if so does that really mean superman is less smart compared with normal human beings?