Rodrigo De Luna Lara
CS7637- Knowledge Based AI: Congitive Systems
Professor Ahsok Goel
Spring 2017
Project 2 Reflection

Please provide a short overview of your approach in Project 1.

For project 1, the selected approach was a mix of visual and verbal. Every part of the problem (A, B, C, etc.) has a frame with attributes for verbal representation (Figure 1) and is loaded as a PIL image object as visual representation.

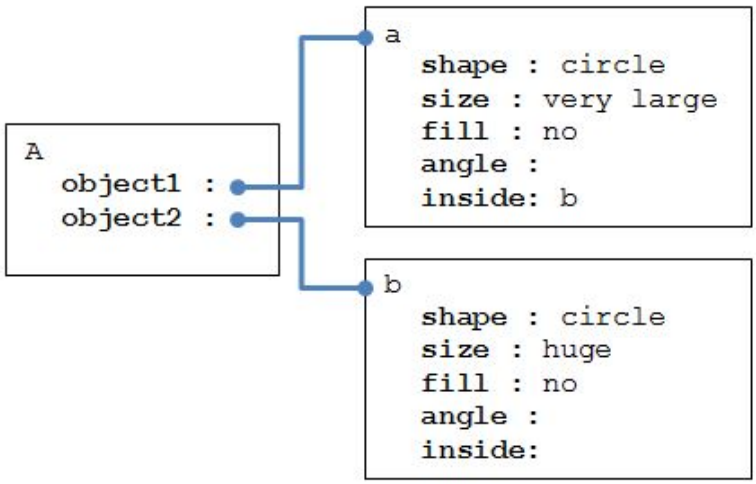


Figure 1. Frame Representation Used for Project 1.

These properties (shape, size, fill, angle, inside which figure the current figure is) are used to determine some basic transformations between parts of the problem, specifically the transformations outlined in Table 1.

Condition	Transformation
if shape changes	'shape-change'
if fill changes	'fill-change'
if angle changes	'rotated'
if object count changes	'added'/'deleted'
if nothing changes	'unchanged'

Table 1. Possible Transformations in Project 1.

Based on these transformations the agent processes the problem in such a way that a prototype answer is generated and tested against the possible answers or the transformation between C and each solution is generated until the transformation matches the solution. This is a very broad way of using semantic networks, the knowledge representations contain the description of the items and the transformations are used to determine the solution.

In some cases one of the transformations is more predominantly used, for example to determine the solution for problems in which the solution is a repetition of the previous frame the agent determines the transformation of C with respect to each possible solution and once it finds that the transformation is "unchanged" it selects that as the response. In other cases the images are processed visually to determine if there were basic transformations like flipping or mirroring.

Based on the results for Project 1, in which for the test problem 2 problems were incorrect, the agent was too specific in some of the processes for determining the solution, perhaps for relying too much in some cases on the calculated transformation, which only considers changes in one of the attributes for the figures. Given that the 2x2 problems are simpler it was deemed that the agent had an adequate performance even with its high specificity.

How was the problem set for Project 2 different from Project 1? What other challenges did you face? Did you change your input representation to incorporate these changes? If yes, how did you make sure that backward compatibility still exists?

Problem Set B has 3 frames from which the transformations can be deducted to attempt to find the answer to the Raven's Matrix. Problem Set C has 8 frames, 5 more, which adds additional difficulty to attempting to solve the problems. In Problem Set B the response could be generated directly from the transformations A-B or A-C. In 3x3 RPM's the transformation to generate the solution can be determined from more combinations, like A-B-C, D-E-F, A-D-F, B-E-G or from A-C, A-F, B-G and A-E. There are four times more possible sources of transformation, which is in itself challenging.

The original architecture to determine the transformation between two frames was kept, but the additional transformation with the third frame is considered additionally in some cases. Some elements had different attributes in the shapes, so the code to determine the feature matrix used for the transformations had to be adapted to consider these additional attributes, which were just additions to the code that didn't modify how the 2x2 RPMs were solved.The frame architecture was maintained as well as the visual representation of the figures, so no significant changes had to be made in the knowledge representations.

In addition, the difficulty of the problems is significantly higher. Some of the transformations can be very easy for a human to understand and use to generate the solution, but programmatically some of them can be really challenging. For example, on problem C-09 it is very easy for a human to see that the figures are moving in opposite directions on the x-axis, but programming an AI agent to detect this kind of transformation can be challenging, with either visual or verbal approaches.
:

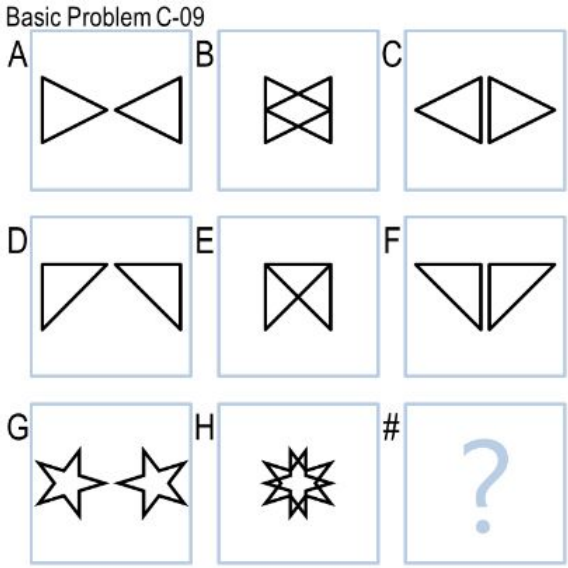


Figure 2. Problem C-09.

Understanding how I as a human would solve that problem helped design the solution in code. Most notably, a human is able to get the solution with "incomplete" information. The middle frames (B, E and H) add no information if one notes that the what is happening between the first and third frames (A-C, D-F) is a sort of image folding, each half of the image is being mirrored. Therefore, an AI agent can be trained to look for patterns between the initial and final state only as a transfer function without requiring it to analyze what is happening in the middle state, which made some solutions simpler.

For example, for problem C-10 the middle states can be ignored and the transformation can be thought of as duplicating the item on the first frame and separating them by a given distance. Using this logic the problems can be reduced from 3x3 RPM's to 2x2 problems with known solution to generate the prototype solution for the 3x3 problem.

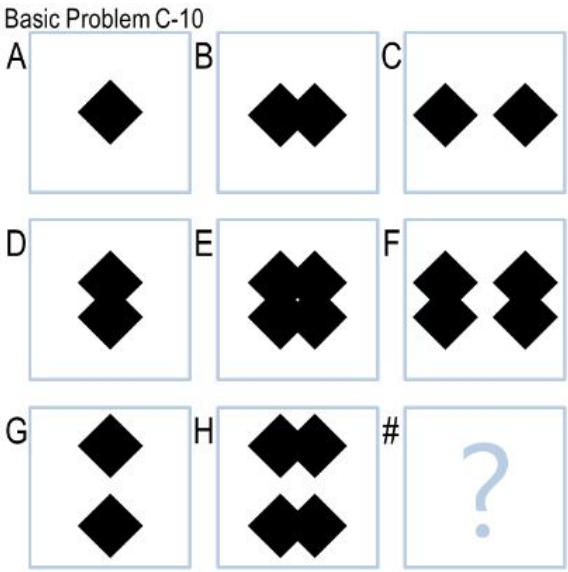


Figure 3. Problem C-10.

How did your agent efficiently try to solve the new challenges without changing its approach for the earlier problem sets? How did you attempt to solve other challenges you might have faced? Did your reasoning approach change? If yes, how?

For this project the approach changed to a mostly visual approach, leveraging more of the methods available in the PIL libraries to perform operations on the images. In Project 1 only some basic operations were made on the images, like mirroring and flipping, in this project it was decided to try to solve all of the available 3x3 problems using visual processing as much as possible.

For each problem the agent was trained, PIL methods were tested to determine the operation the image is subject to between frames to generate the solution. For the most basic problems (like problem C-01 in Figure 4) where the solution is just a repetition of the same frames, the same strategy as in Project 1 was used, checking that the objects in each figure are the same and that the transformations are "unchanged", this uses the verbal representation as outlined in Flowchart 1 in the Flowcharts section.

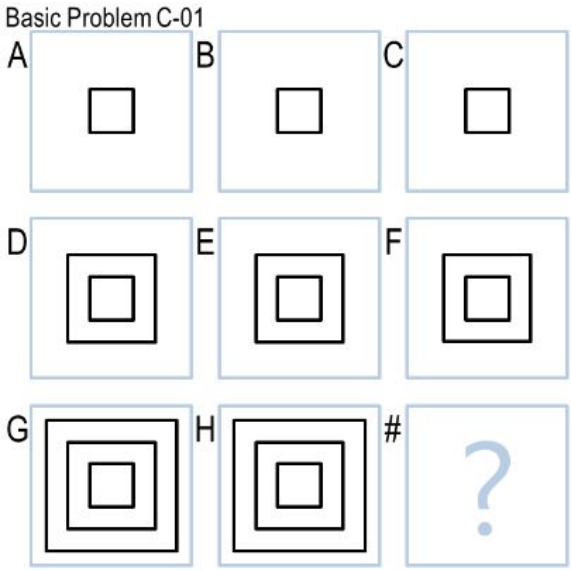


Figure 4. Problem C-01.

The next logic programmed in the agent was for problems in which a shape (the same shape) is being multiplied with every consecutive frame, like in problem C-03 (as shown in Figure 5). In this case the verbal representation is also used for simplicity, the agent follows the procedure in Flowchart 2 to determine the solution.

In both of the previous cases the agent has the flaw that it looks for an exact solution, which in case of the problems it was trained on is adequate, but maybe one of the test problems has the shapes being multiplied and changed at the same time, which the second logic would be unable to solve.

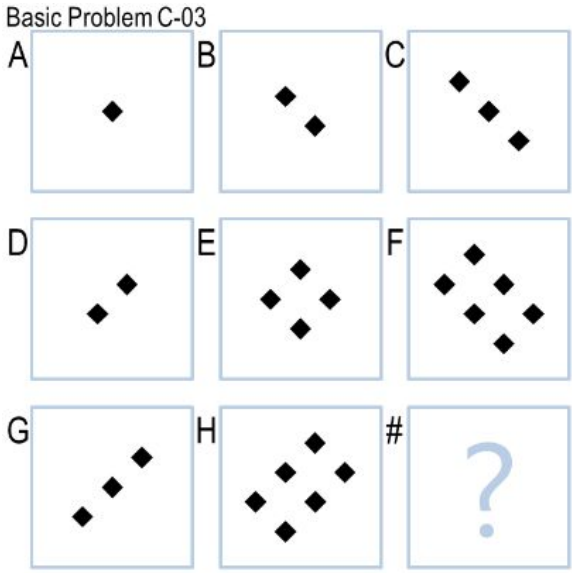


Figure 5. Problem C-04.

The next logic used in this project uses the ImageMath module from PIL to get the mathematical difference of two images, in cases in which between parts of the problem a specific item is added to each element row-wise, like in problem C-12 (Figure 6). In this problem a human can easily identify that in the 3x3 grid between the first and second columns the middle square of the first row is filled in for all three rows, and for the first and second rows the leftmost square of the first row is filled in between the second and third columns.

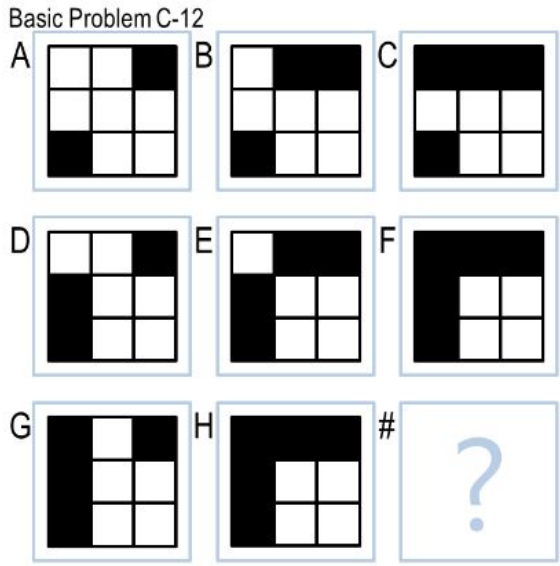


Figure 6. Problem C-12.

Using PIL ImageMath the AI agent can try to determine if the above scenario is happening by getting the difference between the first two columns of each row and comparing them, like shown in Figure 7. The same operation is applied between B and C, as well as E and F, to build the prototype solution from H based on the transformation between these 2 set of frames in the problem.

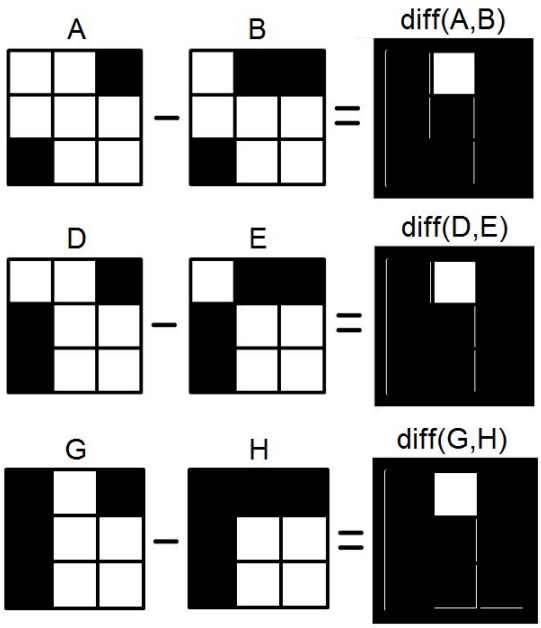


Figure 7. Difference of first two columns in problem C-12.

This is the first visual only implementation for problem solving in the AI agent. The process it follows to determine the solution based on the image differences can be seen in Flowchart 3.

One of the challenges the agent has is the determination of the accuracy between two images, as there can be some noise or jitter in the images, as can be seen in Figure 7. This kind of imperfections in the images can be a source of mistakes by the agent until an adequate measure of accuracy is used. At the moment, the agent uses the root mean square error and a simple matrix similarity score based on the amount of pixels that are different between images only. However, with these limitations the agent was able to successfully solve the problems in the Basic set in a local environment.

The next additional logic implemented is based on problem C-06 (Figure 8), in which a single shape (in this case a square or rectangle which can be thought of as an arbitrary figure with constant width and height) is being enlarged in any direction. In the example figure the shape gets wider between columns or taller between rows. This is easy for a human to do because we can compare the relative size of a figure without requiring to examine it closely and regardless of its location.

The problem was reduced to a simple dimension difference calculation by cropping the figure (removing all the whitespace), and then comparing its dimensions between steps of the problem. So then if B changes its width with respect to A in the same proportion as E-D and H-G, and so does C-B and E-F, then the dimensions of the solution can be calculated from H. The full logic can be seen in Flowchart 4.

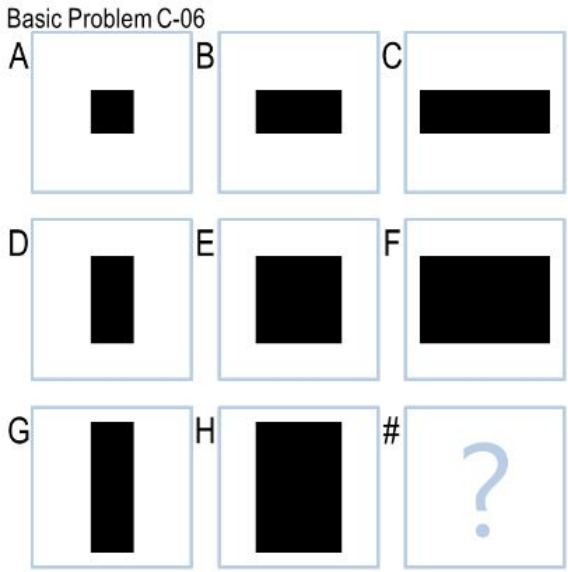


Figure 8. Problem C-06.

The next addition to the logic of the program is just an enhancement from code in Project 1 for cases of image mirroring or flipping, to consider the extra figures in the problem. This is helpful for problems such as Problem C-07 (Figure 8), in which the problem can be reduced to a 2x2 rpm using frames A,C and G or B,C and H. This is a case in which the problem can be reduced by ignoring some of its components if the transformation between the initial and final state can be obtained directly.

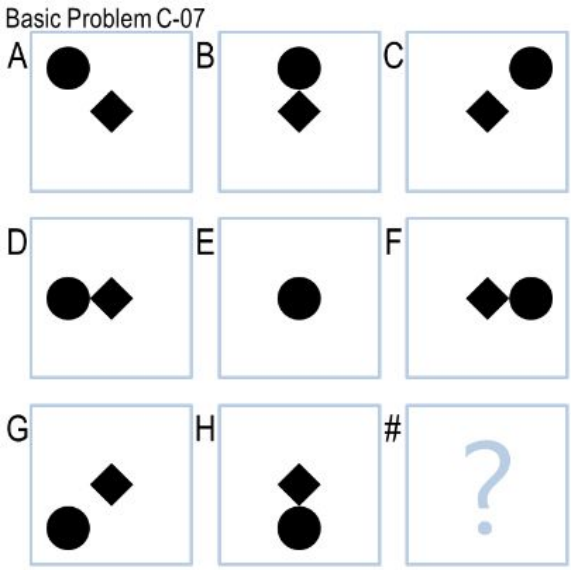


Figure 8. Problem C-07.

The process for solving this kind of problems was already outlined in Project 1, and can be reviewed in Flowchart 5.

The next logic considered in Project 2 is a special case designed for problem C-08 (Figure 9). As a human it is difficult to describe the transfer function A-B-C, but one easy way of seeing what is happening is by looking at the middle frames B and E. It can be seen that between the second and third columns the right part of the image is being mirrored in each case regardless of what is on the left. Taking frames D and E as example, to get the right part of the image the operation D&(!E) can be created (first column in Figure 10), inverted (second column in Figure 10) and added with its mirror to produce the result (third column in Figure 10). The logic the agent follows in this case can be seen in Flowchart 6.

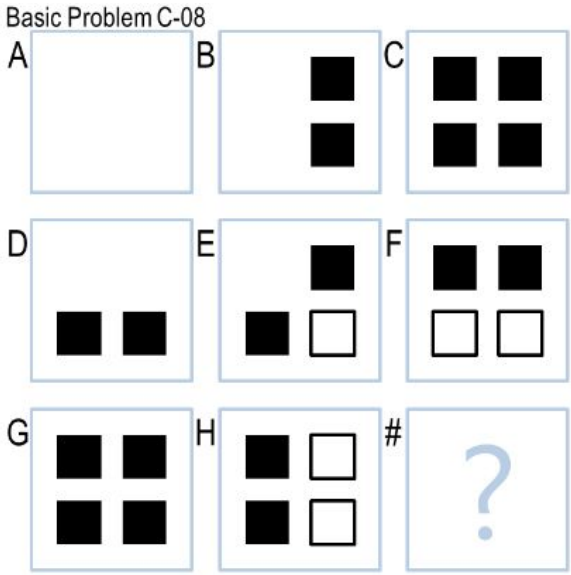


Figure 9. Problem C-08..

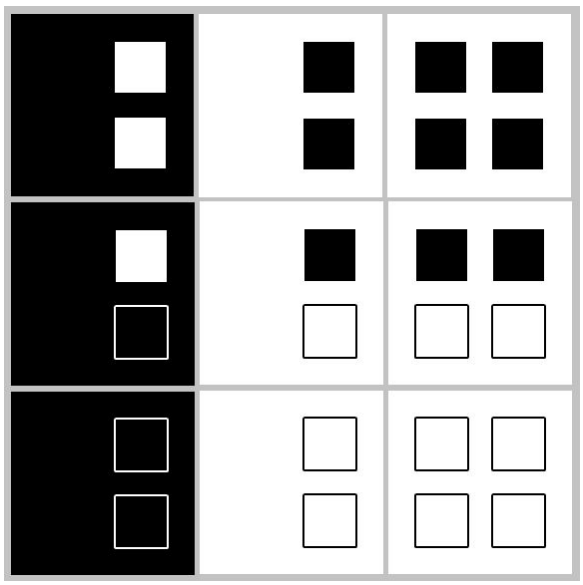


Figure 10. Generation of response.

The next logic is for problems like C-09 (Figure 2). As previously mentioned it's easy for a human to deduct that the shapes are moving in different directions, merging in the middle column to finally point in opposite directions in the last column. To process this visually the middle column was ignored and the first and third columns were cropped of whitespace.

Once having the cropped images, the first column is offset by half of its pixel length using the ImageChops offset operation, which wraps the image depending on the offset magnitude. Offsetting an image by half of its length using this method effectively folds it so that the resulting image is like the mirroring of its halves. Figure 11 shows the resulting problem when the middle column is ignored and the images are cropped to remove any whitespace, while Figure 12 shows the result of offsetting each cropped image by half of its length.

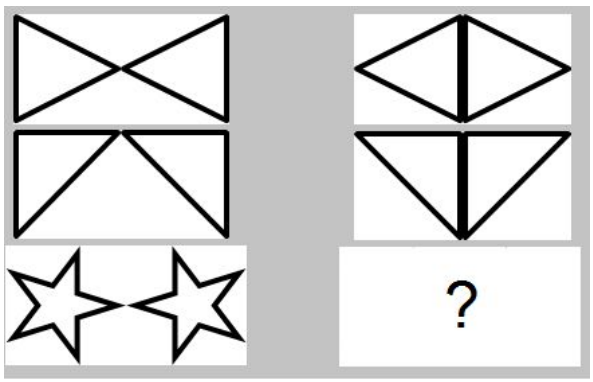


Figure 11. Reduced problem C-09.

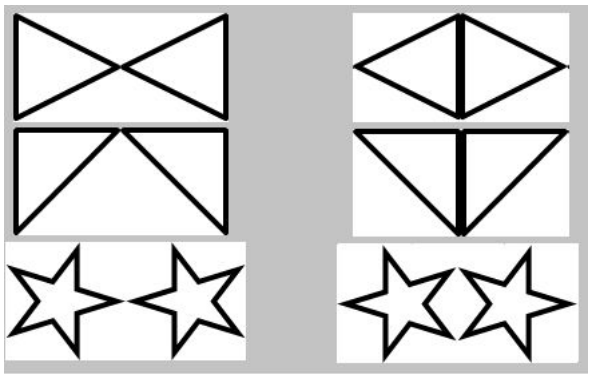


Figure 12. Images generated from offset.

Following this procedure, the response is generated and then tested against the possible solutions, the one that results in lower error is selected as solution, as seen in Flowchart 7.

The final logic implemented for the AI agent is based on problem C-10 (Figure 3). In this case the middle state is also ignored. As said before, a human could see that between the first and third columns the shapes are being duplicated and offset. The AI agent looks for the same logic in the problems, it first crops the images as in the previous logic and then either stacks a copy of itself horizontally or vertically to form a new image, as seen in Figure 13.

Removing the whitespace, the generated stacks correspond to the solution. So the agent checks first if it can generate one of the known frames in the problem by stacking the originating frames, if the relationship is established then it is used to generate the solution and test it against the available solutions.

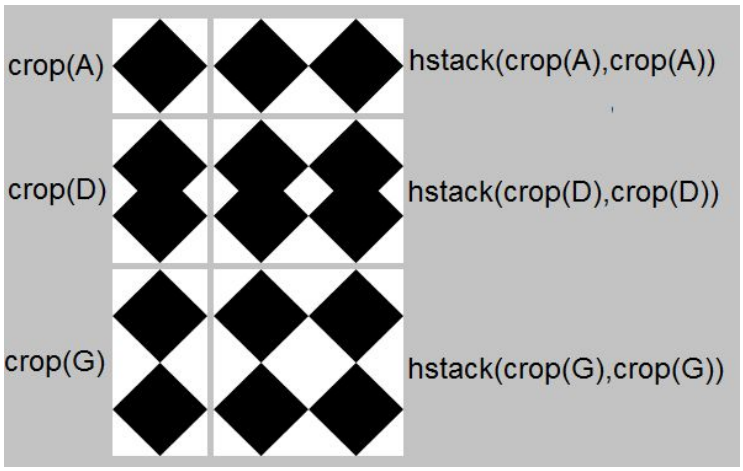


Figure 13. Horizontal stacking for problem C-10.

It can be seen that from all of the explained methods only 2 use the semantic network approach to solve the problem. The rest of the methods use pure visual processing to attempt to solve the problems, which is a significant deviation from the previous project. In which the approach mixed verbal and visual reasoning.

Have you rectified your previous version's mistakes? Are there new mistakes that your agent now makes in the new problem set? Could these mistakes be resolved within your agent's current approach, or are they fundamental problems with the way your agent approaches these problems?

Based on the results in Bonnie, the performance of the agent was affected in the problems for Project 1, it skipped some problems it was able to solve properly before. Being unable to see what the test problems are, the reason why the agent skips the problems is still unknown, so it is a fundamental problem in the design of the agent, which is due to some difference in the architecture or execution of the script, as the performance of the agent locally wasn't affected at all by the changes made in the design.

The performance of the agent for Basic Problems C in a local environment is of 100% accuracy in the results, having no mistakes. This differs sometimes with the results when the code is submitted, so there may be a design flaw in the use of dictionaries that was addressed in Piazza. However, even when attempting to solve this issue the performance keeps being inconsistent, which may point at a flaw in the way the solution is selected.

Perhaps some of the mistakes the agent makes is due to inadequate programming in Python, which I just started to use in this class and I'm not very experienced in some of the harder to catch things that may not necessarily result in a warning or error but could be affecting the performance of my agent. To solve this issues the test problems would need to be available so that I could debug my code against them directly, or perhaps my development environment should match that of the system running the project submissions.

Using the new changes what was the performance of the agent? Did you use the same evaluation metrics are earlier? What were the new ones, if any? Did your agent generalise well enough from earlier without the new changes?

Table 2 outlines the accuracy of the agent in Project 1 and in Project 2 in the local environment and in the best submission.

	Basic Problems B	Test Problems B	Basic Problems C	Test Problems C
Project 1	12 correct	10 correct 2 incorrect	NA	NA
Project 2 (Bonnie)	12 correct	9 correct 3 skipped	11 correct 1 skipped	8 correct 2 incorrect 2 skipped
Project 2 (local)	12 correct	NA	12 correct	NA

Table 2. Performance Scores for Project 2.

In project 1 the overall accuracy was considered without making the distinction between skipped and incorrect problems. In project 1 the issue seemed to be with the accuracy in determining the solution or having some methods too general with respect to the others. The results for project 2 show that the changes introduced in the agent reduced its generalization and increased its specificity, which is an inherent issue with the design of the agent.

Without being able to look at the test problems, it can be concluded that the methods are too specific and not very adaptable, and that some of the methods approximate some of the test problems but are unable to produce an adequate solution or are unable to select the best one if there are similar shapes for the response. The factor of the difference in environments that could be the cause of having different results locally versus in the server is also important, as it is severely affecting the performance of the agent.

Having a mostly visual approach simplifies some of the logic of the AI agent but doesn't make it very robust when facing unseen images or when one problem doesn't comply with all of the assumptions that were made for some of the methods of the agent. For example, removing all whitespace from images helps solve some problems but in some unseen ones it could be an essential part of the answer and not so easily disposable.

Please provide an explanation of how your methods/components/ideas in your agent's design are/might be similar to (or can be related) to specific KBAI methods discussed in class

The main topics in KBAI that support the development of this project are logic, planning and generate and test. Logic and planning are the backbone of most of the visual methods implemented in this project, as each problem is tested against a set of logic methods which test progressively its similarity with a set goal which represents the final step in a sequence. For example, for the case in which something is added incrementally between frames, the agent tests the cases against several states with known properties, if between A and B something is added, it tests that D and E are able to reach the same state in F, and does so in a set of steps instead of generating the solution instantaneously.

If one case is unable to reach a state in of the implementations the agent drops the current method and tries another one, until the problem matches the states of one of the methods and the goal is able to be generated from the available data. Logic is used to ensure compatibility between variations of the same problem. For example, an image can be either mirrored or flipped to produce one of the states in a method, or it could be subject to a logical operation with another image to reach the goal state, logic helps set up the required preconditions and postconditions for each method.

Generate and test is used at almost every method, if a problem matches the signature of one of the methods the AI agent has at its disposal, then an answer is generated from the available data and tested against the possible solutions. However, this has the disadvantage that when dealing with image processing methods, there can be noise or jitter introduced to the image and the determination of the solution is largely dependent on the accuracy score used to compare images. The agent uses the root mean square difference[1] between images as accuracy score, but this is more difficult to understand and use than scores such as the Structural Similarity Index[2], which has a simpler interpretation although it is significantly harder to calculate.

Planning and logic are very powerful tools for design of AI methods because they are very structured and are natural in programming. Each method can be made to have “tollgates” for states, which allow the AI agent to determine if the current method is appropriate for attempting to solve a problem. Logic also provides a strong foundation to validate the methods and help the AI agent decide which one is applicable to each problem.

What does the design and performance of your agent tell us about human cognition? Do you think your agent is more human like because of the new changes? What new insights do you have now, after having seen more problems for the agent to solve? Has its reasoning or problem solving become more or less human like? What metrics do you use to arrive at this conclusion?

The agent for this problem is more human like than the agent for the previous project. The Raven’s Progressive Matrices are naturally solved visually by humans, any verbal method would be unnatural for any person taking a RPM test. Therefore, having a more visual approach brings the agent closer to human cognition.

After seeing more problems for the agent to solve, it is evident that humans have a very complex processing system and are able to detect patterns from either complete or incomplete information very quickly, without much complex processing. An AI agent on the other hand has a harder time finding

patterns unless it's given some form of guideline, and visual processing is more similar to the cognitive process humans follow to solve the problem.

For example, for Problem C-05 a human can easily see that for each consecutive frame a dot is added to the right of the point of the star with the last point clockwise, the AI agent had to be programmed to figure out what is changing between frames and to produce an answer based on G and H, but this is done on a Master's level class using some not very basic Python programming. A kid could probably solve the problem effortlessly.

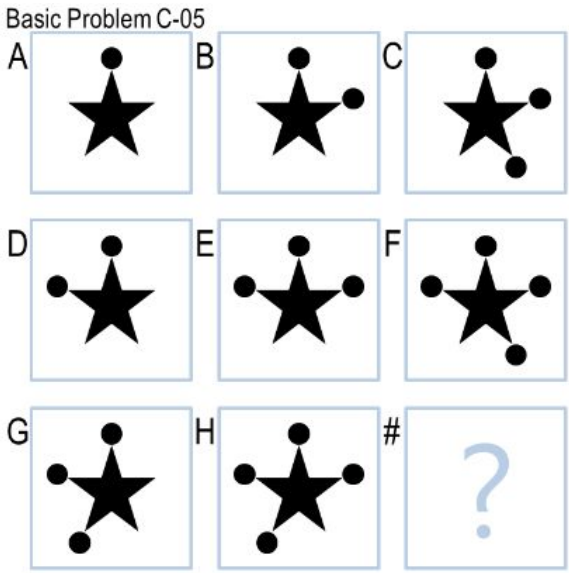


Figure 14. Problem C-05.

The modifications done to the agent showcase how complex human cognition is, but if a problem is abstracted to a simpler problem with focus on pattern recognition, an AI agent can be made to approximate human cognition on a very basic level, it has become more human like because the logic and planning put into every method were specifically programmed taking into account what I as a human think and try to see to solve the problems, and so the agent is based on my own cognitive process (which certainly introduces a bias and may be a cause for mistakes in the agent).

Finally, no specific metrics were used to determine how close the agent is to human cognition, but the overall design changed to a more visual approach, which inherently is more human-like, as said before any human would much prefer a visual approach than a verbal approach because our brains are very good at finding patterns. Just the overall approach of the agent changed and so it became more alike human cognition.

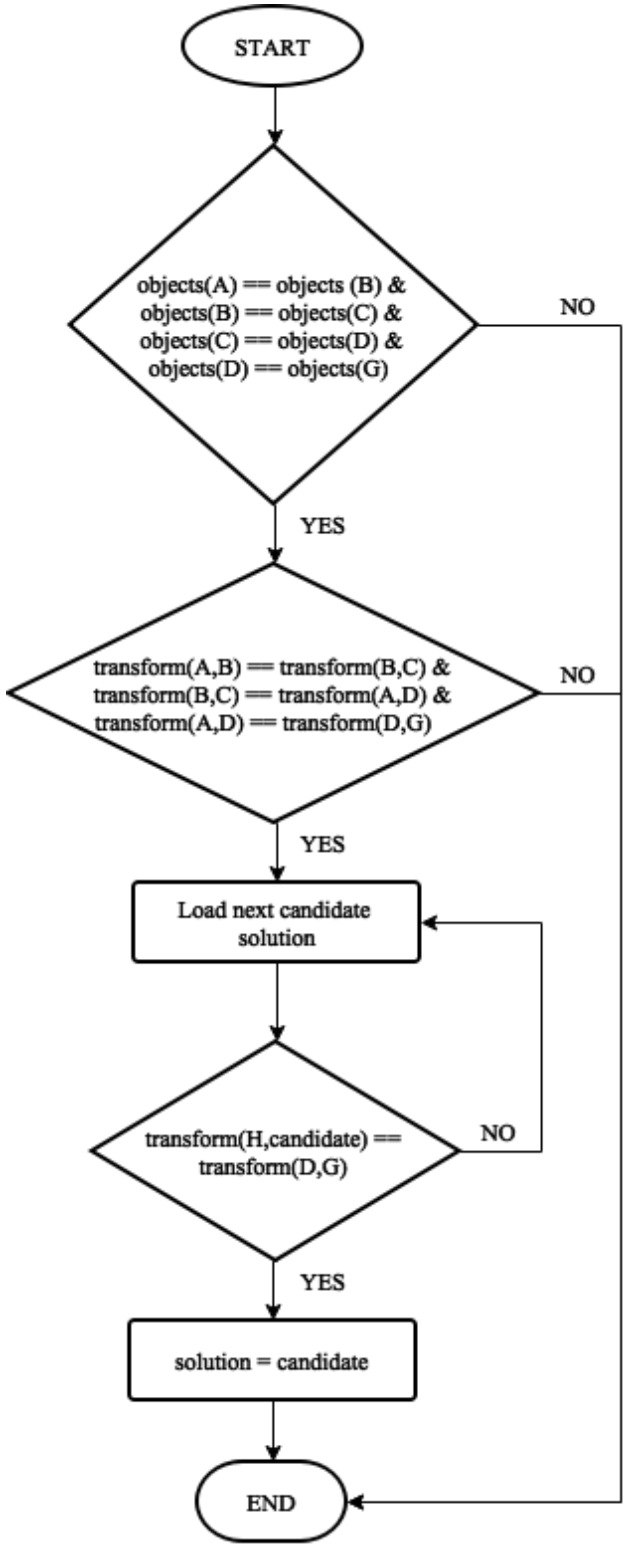
References

[1] Lundh, F. (1997) Comparing two images. Retrieved from http://effbot.org/zone/pil-comparing-images.htm on March 19th 2017

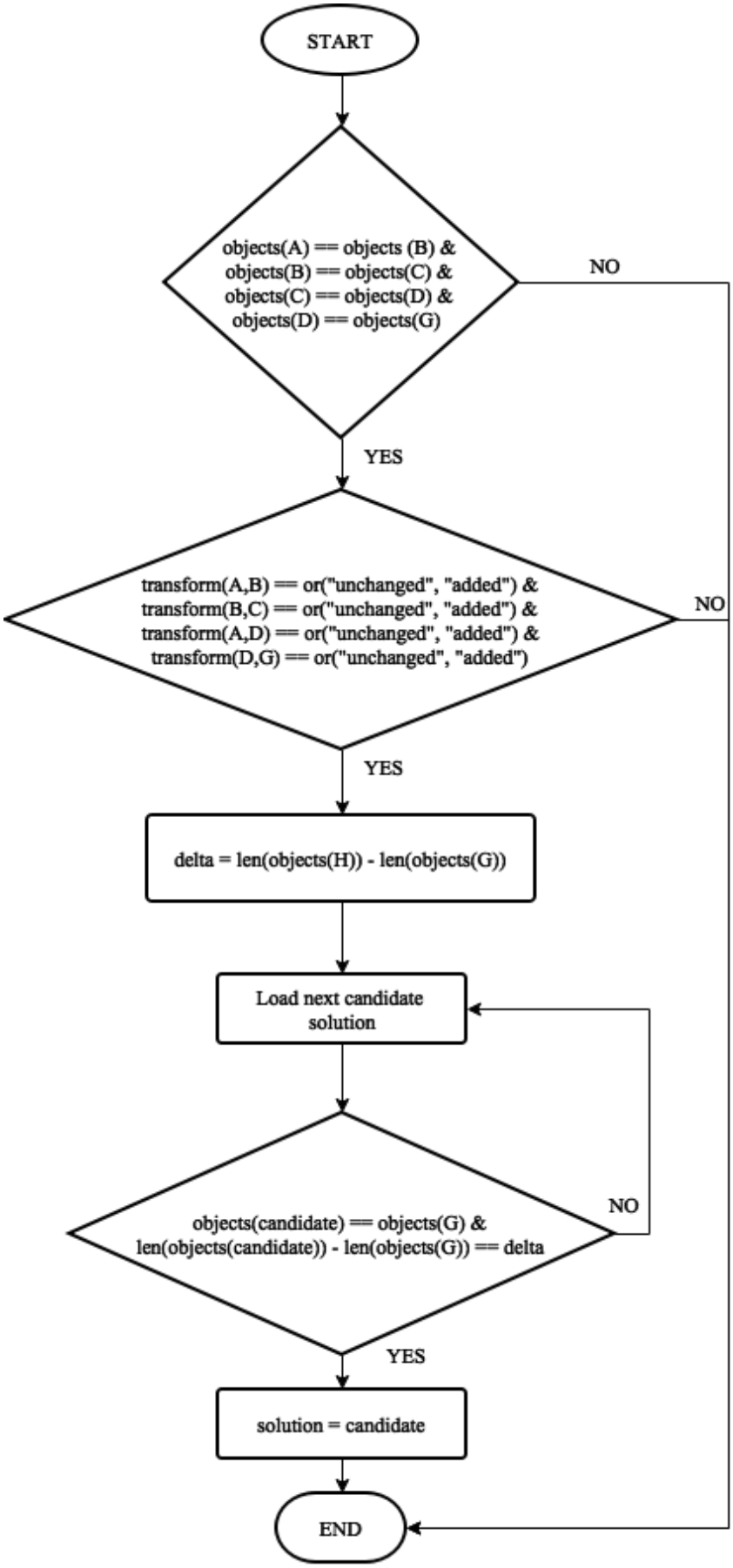
[2] Sampat, M., Wang, Z., Gupta, S., Bovik, A., & Markey, M. (2009). Complex Wavelet Structural Similarity: A New Image Similarity Index. IEEE Transactions on Image Processing, 18(11), 2385-2401. doi:10.1109/tip.2009.2025923

[3] Clark, A. (2017). Pillow (PIL Fork) Reference. Retrieved from http://pillow.readthedocs.io/en/4.0.x/reference/index.html on March 19th 2017.

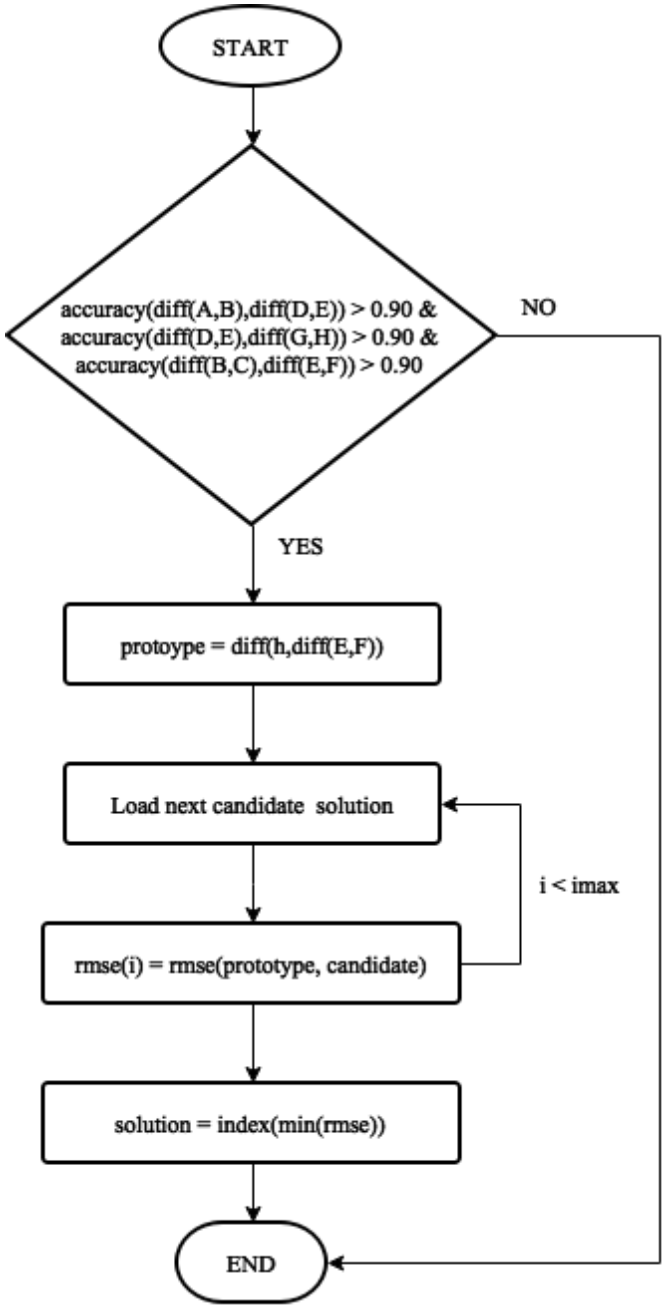
Flowchart 1: Logic for “unchanged” cases



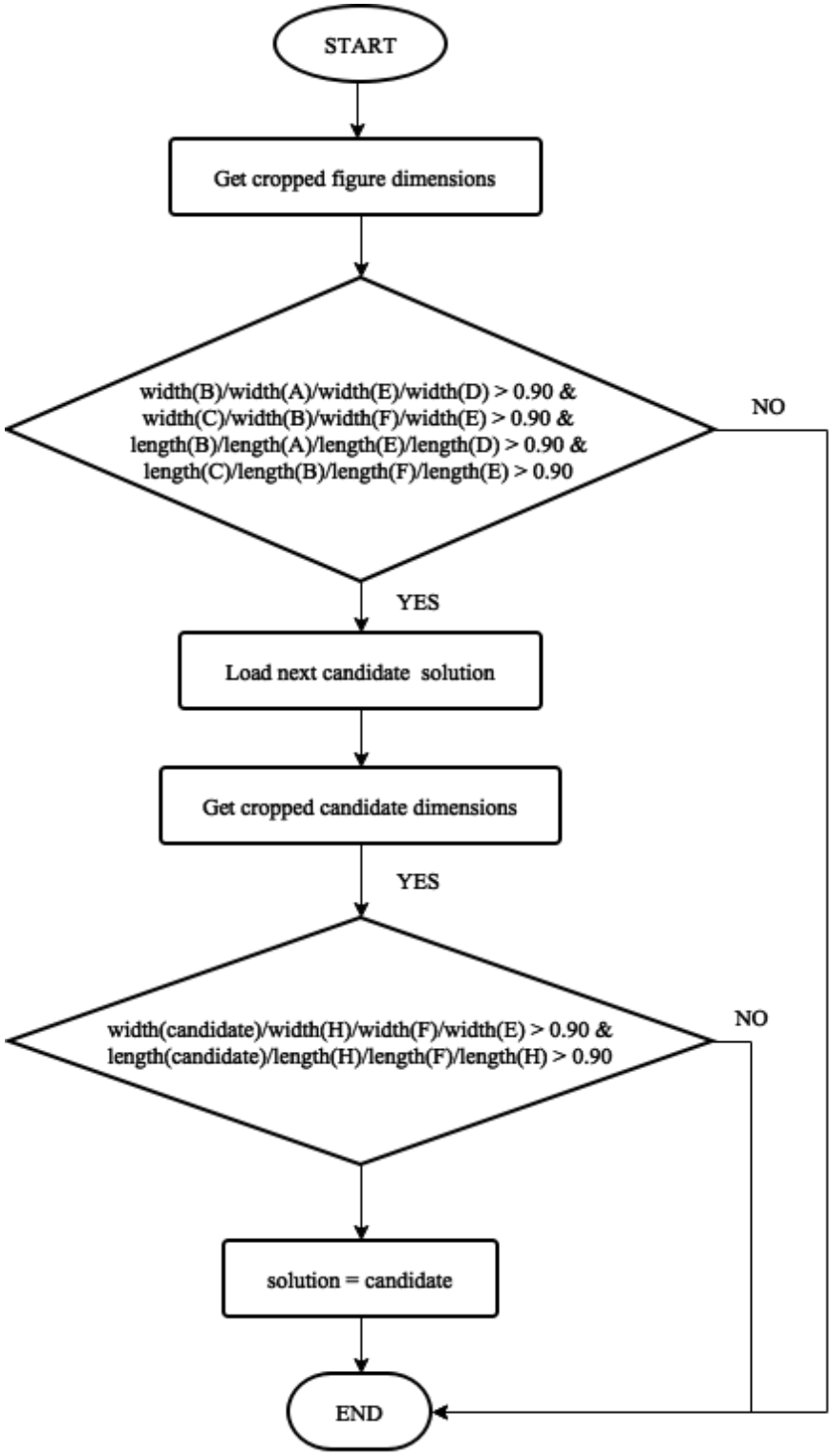
Flowchart 1: Logic for “elements added” cases



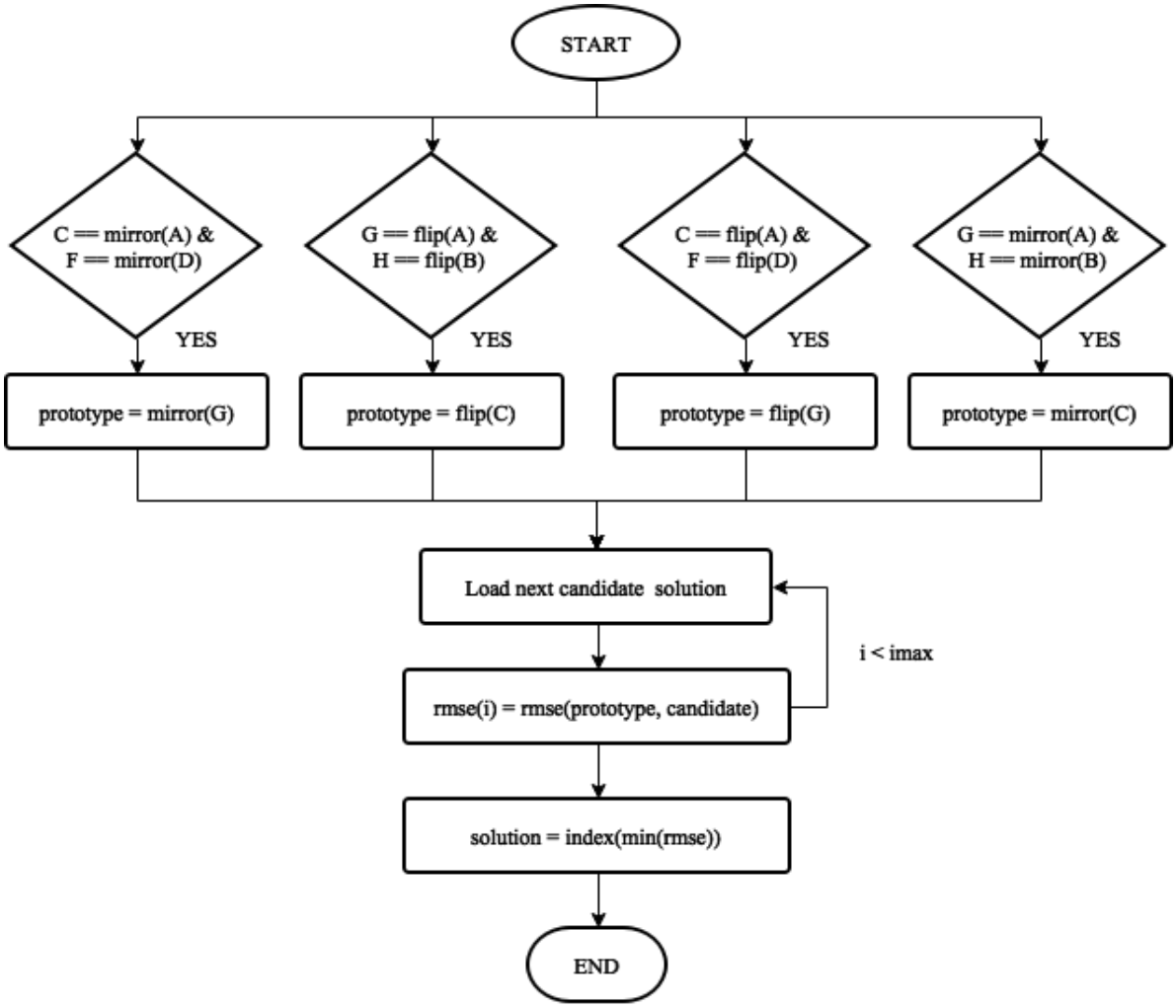
Flowchart 3: Logic for "image difference" cases



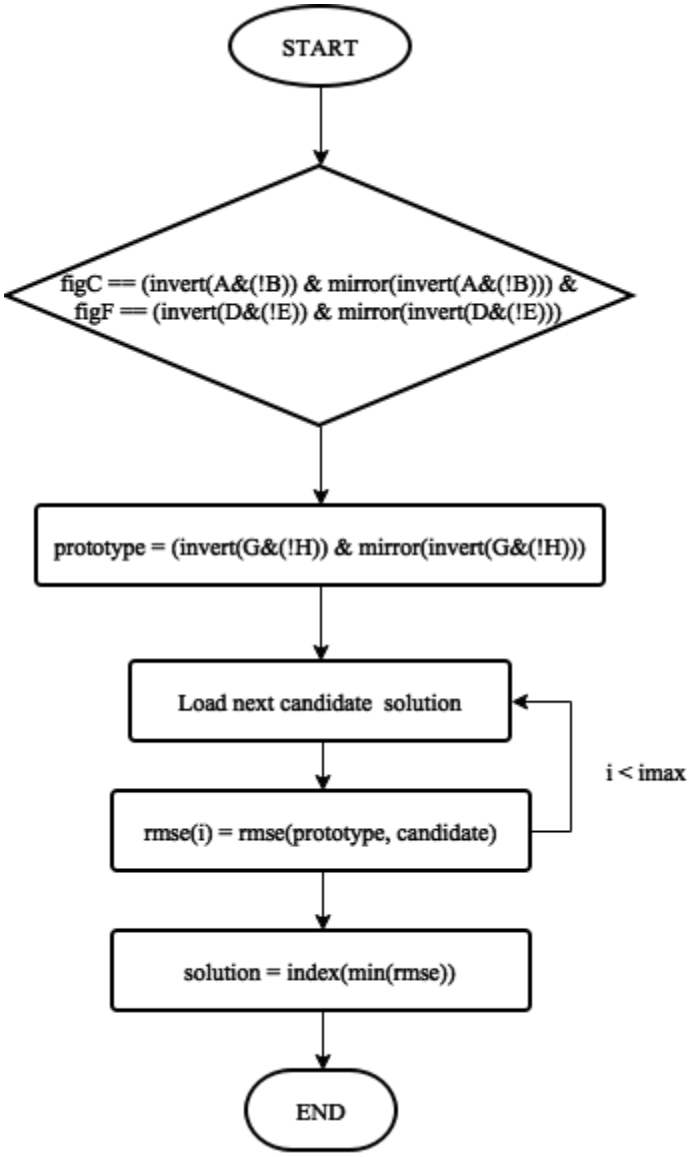
Flowchart 4: Logic for “shape resize” cases



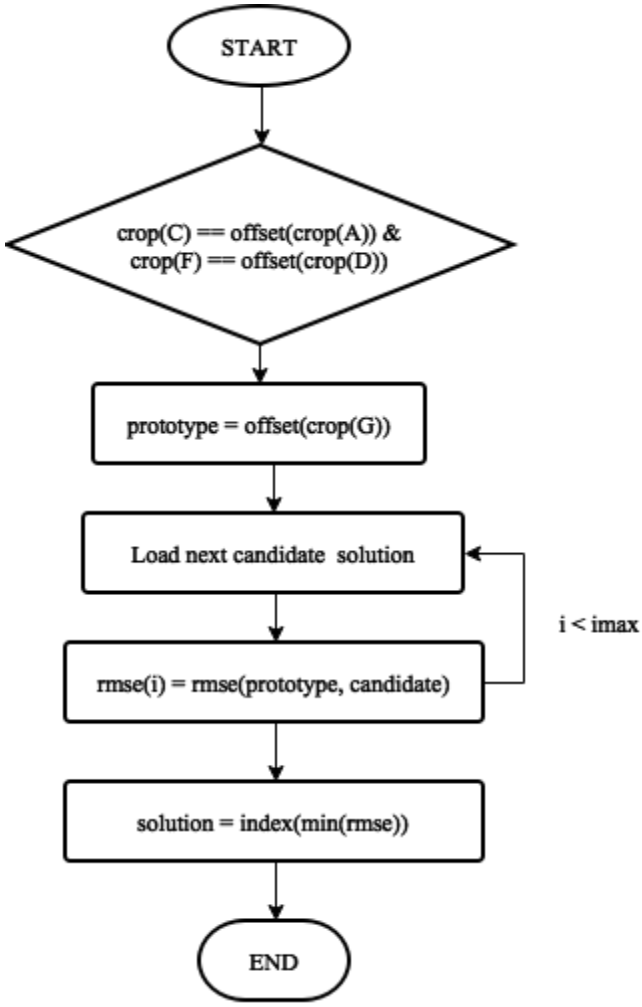
Flowchart 5: Logic for "flipping/mirroring" cases



Flowchart 6: Logic for "mirror right side of frame" cases



Flowchart 7: Logic for “image folding by offset” cases



Flowchart 8: Logic for “image stacking” cases