Vinodh Kumar Sunkara, Yahoo Inc.

Before getting into details of my agent, I would like to quote the importance of architecture first. I strongly feel that architecture is very important before building any agent. More we teach, more the agents learn. But it's important for AI agents to learn in a correct way to apply learning on unknown problems. We should build the agents in a way that all the existing learning is applied correctly on a new problem to come up with solution. This is achieved only through a systematic architecture for agent implementation. A bad architecture can unlearn old things with new rules or misguide agents to come up with solutions lacking confidence.

For 2x2 matrics, agents have to take into consideration horizontal and vertical ratios. A:B matches with C:D and A:C matches with B:D. I've added different rules on different levels to be applied horizontally and vertically both to come up with a common solution. I've a level1 set of rules with comparison between two figures to generate the missing figure. These rules are applied one by one horizontally and vertically in all combinations to come up with a common figure. The generated figure is provided to a tester which matches for a correct option out of six figures in answer choices. I've a level2 set of rules which are applied if level1 rules can't come up with a solution. The level2 rules contain a concept applied on all figures together. They have a piece of concept using three figures A, B, C and tester has a much better role here to find an option from answer choices that can complete the concept.

An example of a level2 rule is my bit string pattern completion rule. For problem B-04, I've assigned a bit 1/0 per quadrant in each figure. A fully white quadrant takes a bit 1 and 0 otherwise. This gives me 0001, 0010 and 0100 respectively for A, B, C figures. I finish the concept by generating a bit string 1000 for missing figure and pick right option by generating bit strings, option by option from answer choices to match up with "1000". An extension of this level2 rule is to add weights. Problem B-07 has similar figures but the non-white quadrants are either the circular arcs or filled circular sections. I take the ratio of non zero pixels (using numpy lib) from A:B or A:C, to obtain the same with missing figure from options.

The level1 rules are mostly a combination of comparing figures, rotations, reflections, multiplying figures, flipping figures left to right or top to bottom, finding the shade type - if shade spreads from top to bottom or bottom to top, finding the diagonals - left to right or right to left, comparing no. of vertices of inner figures and checking inner fill in different shapes, finding quadrant wise ratios between figures horizontally and vertically. As a future item, I would like to separate out the rules which involve the double for-loop to read image pixels and create them as level3 rules which are applied, only if level1 and level2 rules with PIL operations and numpy operations on figures can't derive solution.

The risks I've taken are by involving new methods like weighted bit string as described above and comparing the quadrant ratios. Main risk I took was not to loop over all pixels, which would increase the computation time. I've succeeded with my risks. I had to spend more hours in finding different PIL library methods and numpy functions, but got them all very handy in building a very generic agent that could already solve four raven problems along with all basic and most of the test problems for project1.

There is some possibility of incorrect selection. Main reason is because I haven't incorporated a way for my rule to pick multiple options and use other rules to filter out of the selection. At a time, only one rule can operate on selection with either horizontal/vertical ratio check, and another rule to generate figure matching vertical/horizontal ratio. A future add-on could be to let my rules pick multiple options if they can and use other rules to filter a single output. However, it's working great with what I have so far, so good.

Currently, my agent is very generic. This has been my major goal when I started implementing the agent. I have put very less focus on getting answers for 12 basic problems. My main goal was to build a generic agent without hard-coding any numbers or logics. I've achieved it all right so far and looking forward to extend it with many novel concepts for future projects. Running time for current project isn't very optimized. Main reason is because I'm running all combinations of my ruleset for horizontal and vertical ratio justification for every problem. This makes my agent very generic and confident in solving problems, but it does take a bit longer time than it could have been, had I hard-coded the logic combinations in sequential manner. I'll see fruits of my method on a long run and I'm confident about it. I'll also propose efficient algorithms to decide the order in which rules are applied as I add more rules.

I'm solely going on visual representations. Few rules directly manipulate images and a few find it easy to turn image into quadrants and generate bit strings. It's more generic to deal with the bit strings which care less about actual figure shape and size. I'm not looking at any verbal representation. My agent performs almost the way I do. I try to see if I can generate my missing figure by validating the ABCD ratios. If I fail in it, I try to fit in options for right pick. My agent works the same way. The only limitation my agent has is it's insufficient learning currently. I still have to make it more generic by generalizing many scenarios which I haven't come across so far in this 12 problems. I should also make the upcoming rules very generic, tight and confident that the existing behavior would only be enhanced but not hampered.