Dustin Jones

Ashok Goel

CS-7637

2016-09-04

Semantic Networks and Generate & Test: Better Together

Raven's Progressive Matrices (RPM) are a challenging set of problems to solve not only algorithmically, but also by humans using intuition and reason. This set of problems are particularly unique in knowledge tests; they are purely visual and require no external knowledge of language, science, or arts to complete. When humans complete these problems, they frequently do so without the aid of defined algorithms. Over time, several methods that humans use to solve them innately have been identified and defined; of those methods, semantic networks (SN), along with generate & test (GT) have been demonstrated to be some of the most 'human' in their behavior. Given that humans, when solving RPM without the aid of defined algorithms, use a combination of methods to solve these problems, taking the same approach algorithmically may yield the most human-like solutions.

Semantic Networks

Semantic networks are a method of diagramming a scenario or problem; breaking it down into its component entities, explicitly defining all relations between the entities. When used to solve RPM, SN greatly aids computability by enabling the system to identify, categorize, and link behaviors and transformations of entities. SN allow not only for solving of RPM, but when a perfect solution is unable to be determined a 'best guess' can be discerned by adding metrics to gauge transformations in a human capacity.

Building an initial lexicon or library of potential entities, their properties, and the relationships between entities is crucial to designing an agent that utilizes SN to solve RPM. By building a lexicon an algorithm is able to identify known entities (e.g. triangle, circle, square), and also to create descriptors of unknown entities for use in classification. The second major step is identifying the relationships between entities within an RPM after the entities within them have been identified. By identifying the properties of both the entities and their relationships, they can then be compared to another set of entities in a computable fashion.

After building a model of one set of entities and their relationships, and another set of entities and their relationships inferences can be made upon how the first set was transformed into the second set in a 1x2 or 2x2 RPM. However, when looking at 3x3 RPM some relationships based on sequences or mathematical operations may fail to be discerned by building a SN. Building a library of potential transformations (e.g. translation, rotation, skew), and how they can be applied to entities in a functional, repeatable manor is the third major step in building an agent that can solve an RPM.

Generate & Test

Generate & test alone is insufficient for solving open-ended problems like RPM. A core tenet of GT is producing all possible solutions (i.e. transformations) and then testing those possible solutions to see if they are successful (i.e. a match exists in the listed problem answers). The challenge here is that there are an infinite number of transformations that can be performed on a given RPM set. While the total number of transformations can be artificially limited (e.g. perform a maximum of 3 transformations to a given set), it is not a pure GT methodology.

Approaching the problem from the reverse direction can potentially yield better results. A significant advantage with RPM over other types of problems is that all of the potential solutions are already given. By 'generating' each solution (i.e. picking it from the solutions list), it can minimally be tested against the matrix to determine if it is a viable answer or should be thrown out altogether.

Combining Semantic Networks and Generate & Test

While the potential exists for either semantic networks or generate & test alone to arrive at solutions given unlimited time and resources, combining the practices for both most certainly enhances the likelihood of success and reduces inefficiencies in both methods when used alone.

An agent using both SN and GT can be built in several steps. The first step always being to identify and classify entities, and sufficiently describe the properties of unknown entities such that they may be identified uniquely. Once all entities have been classified, GT is able to build models of potential mappings from one set to another. For example, in Fig. 1, GT might identify 2 potential models from A -> C: the circle in A matched with the triangle in C, the square with the circle, and the dot with the dot; or the circle matched with the circle, the square with the triangle, and the dot with the dot. It might identify 2 potential models: from A -> B, the dot was deleted and the inner square expanded and moved; or the square was deleted and the dot transformed into a square.

Given these potential models, SN can build relationships between the models to describe the changes and analogies. Once the full set of entities (and their properties), and relationships have been mapped, GT can then be used to apply the library of transformations, in ranked order by the metrics of SN (e.g. moved/expanded > transformed from dot to square in Fig. 1). The

generator might apply a limiter to its process to calculate if a proposed set of transformations exceed a maximum ‘cost’ preventing combinatorial explosion. Once the full set of transformations has been calculated, the tester can compare the SN model of the result to compare for matches in the provided answers, again ranking by metrics to determine the best possible solution.

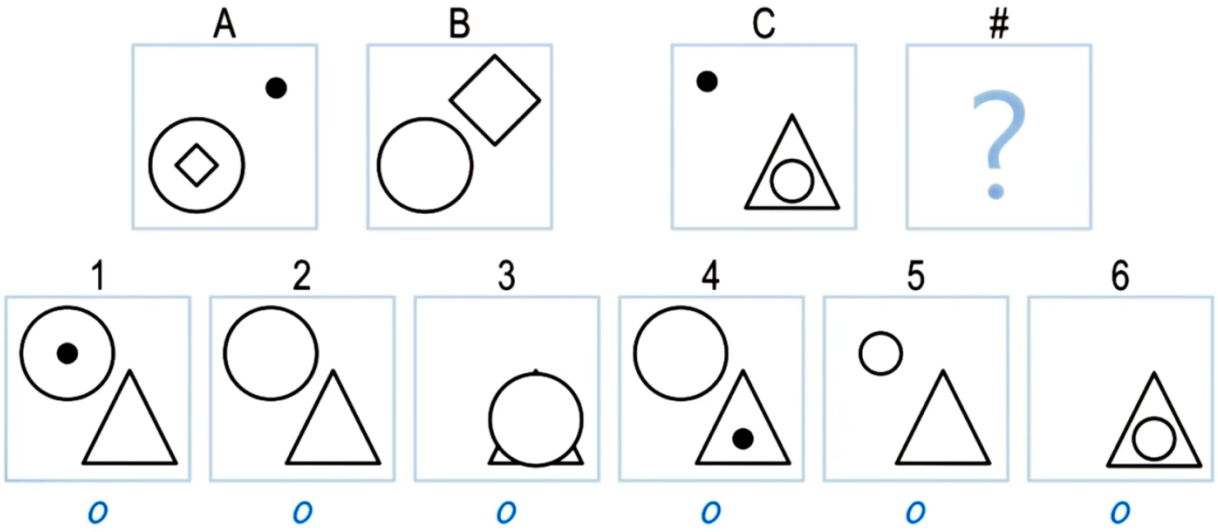

Fig. 1. Example image of 2x1 RPM

Solving a 3x3 RPM may prove challenging even with these methods combined. While they are a powerful tool, SN are still constrained by their limited knowledge, such as the library of transformations that it has available, or the logic for identifying properties of entities. Relationships in a SN don't necessarily lend themselves to inferring meaning between multiple entities where sequences or sets which are graphically unrelated are involved (Fig. 2), only pairs of entities. The model could potentially be improved by examining relationships between entities independent of row and columns.

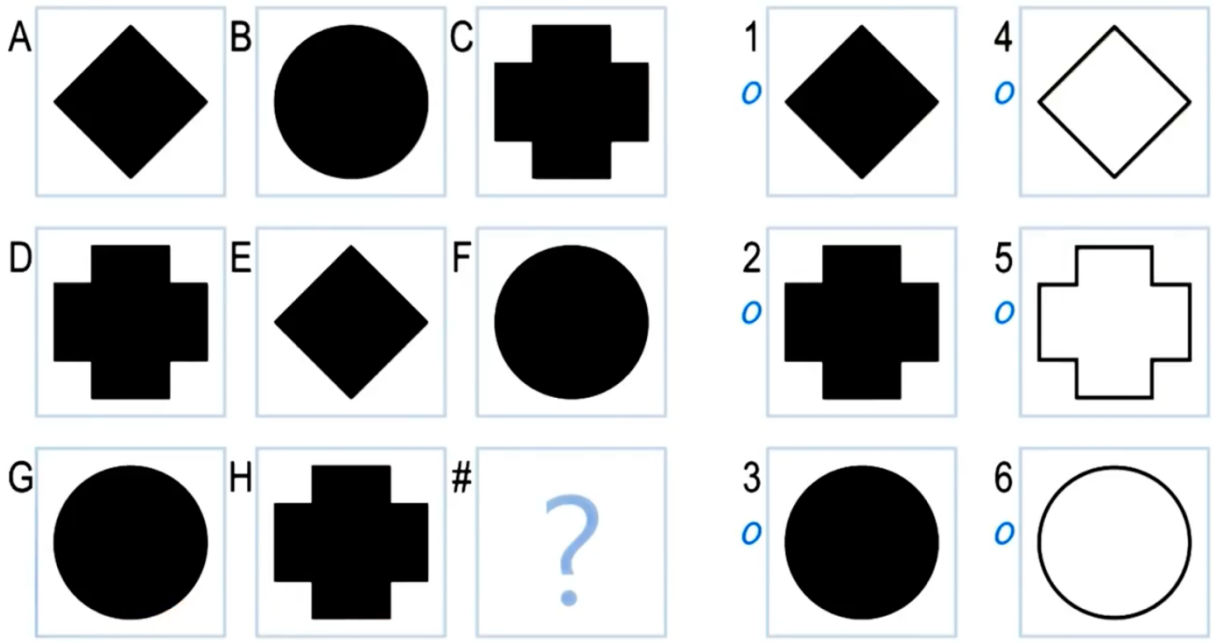

Fig. 2. Example of 3x3 matrix without transformational relationships

Works Cited

Nicola, Antonio De, Michele Missikoff, and Roberto Navigli. "A Software Engineering Approach to Ontology Building." *Information Systems* 34.2 (2009): 258-75. Web. 2 Sept. 2016.

Winston, Patrick Henry. *Artificial Intelligence*. 3rd ed. Reading, MA: Addison-Wesley, 1993. Web. 2 Sept. 2016.

Goel, Ashok, and David Joyner. "2x1 Matrices II - Georgia Tech - KBAI: Part1." *YouTube*. Udacity, 23 Feb. 2015. Web. 02 Sept. 2016.

Goel, Ashok, and David Joyner. "3x3 Raven's Progressive Matrix I." *YouTube*. Udacity, 23 Feb. 2015. Web. 02 Sept. 2016.