Strategies for Visuospatial Reasoning: Experiments in Sufficiency and Diversity

James Ainooson JAMES.AINOOSON@VANDERBILT.EDU
Joel Michelson JOEL.P.MICHELSON@VANDERBILT.EDU
Deepayan Sanyal DEEPAYAN.SANYAL@VANDERBILT.EDU
Joshua H. Palmer JOSHUA.H.PALMER@VANDERBILT.EDU
Maithilee Kunda MKUNDA@VANDERBILT.EDU
Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN USA

Abstract

In this paper, we present the Visuospatial Reasoning Environment for Experimentation (VREE). VREE provides a simulated environment where intelligent agents interact with virtual objects while solving different visuospatial reasoning tasks. This paper shows how VREE is valuable for studying the sufficiency of visual imagery approaches for a large number of visuospatial reasoning tasks as well as how diverse strategies can be represented and studied within a single task. We present results from computational experiments using VREE on the block design task and on numerous subtests from the Leiter-R test battery on nonverbal intelligence.

1. Introduction

When a person is faced with a novel problem or task, we often, and seemingly effortlessly, can form a strategy for solving it. For visuospatial tasks in particular, our strategies often involve sophisticated combinations of gaze shifts, memory storage and retrieval operations, comparisons, mental transformations, etc. (Tversky, 2005; Land & Tatler, 2009). Strategies formed for exactly the same problem are likely to differ across individuals, and some of strategies might be equally effective, while some might be far more effective or efficient than others. And, whether a strategy fails or succeeds, knowledge gained from the attempt can be used to form new and often improved strategies.

Take a look at the problem shown in Figure **??**. Your goal is to select images from the options labeled “A” through “F” that correctly complete the dashed slots labeled “1” and “2”. Without any description of what is expected, which of the choices (“A” through “F”) will you make? What made you select the strategy for deciding on your choice?

The “correct” interpretation of this problem is that the images on the top row represent rules that define some binary operations over images, and these operations should be chained to solve both problems on the bottom. However, even if the images for the rules on top are totally disregarded, it turns out that correct final solutions can also be obtained just by considering which answer choice represent pixel-wise set combinations of the input images. The solution for the slot labeled 1 can be found by performing the NAND binary operation on all images pointing to the “1" slot, and the

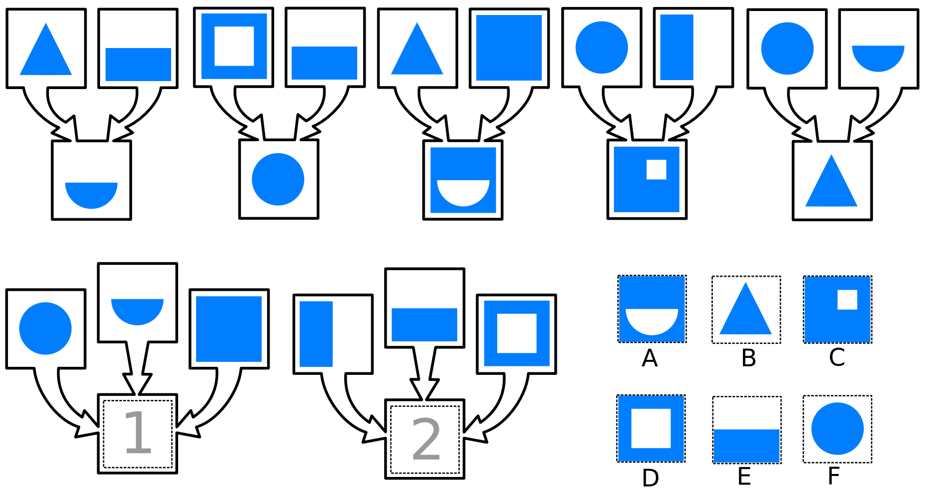

Figure 1. A visual reasoning task that requires you to find the image patches from the ones labeled “A" through “F" that rightfully fit the slots labeled “1" and “2". This sample task has been adapted from one of the items in the Visual Coding subtest of the Leiter International Performance Scale-Revised.

solution for slot 2 can similarly be found by performing the OR operation on all its input images. While this is almost certainly not the strategy intended by the problem's author, it nevertheless produces the correct answers, at least on this problem. [1]

How can we represent and study such strategy variations in computational agents? In this paper, we present the Visuospatial Reasoning Environment for Experimentation (VREE). VREE provides a simulated environment where intelligent agents interact with virtual objects while solving different visuospatial reasoning tasks. This paper shows how VREE is valuable for studying the sufficiency of visual imagery approaches for a large number of visuospatial reasoning tasks as well as how diverse strategies can be represented and studied for a single task. Our contributions are as follows:

1. We first demonstrate, in a proof-of-concept experiment, the sufficiency of a relatively small number of visuospatial operations for solving 17 subtests from the Leiter International Performance Scale Revised (Leiter-R) (Roid & Miller, 2011), a standardized test of human nonverbal intelligence. Essentially, through this experiment, we establish the foundation for a domain-specific language for expressing strategies that use imagery-based operations to perform a variety of visuospatial reasoning tasks.
2. We embed this language within the Visuospatial Reasoning Environment for Experimentation (VREE), which additionally provides 1) parameterized constraints related to an agent's embodiment, e.g., visual gaze that can be sequentially directed to regions of the external environment, and motor actions that must be enacted to effect change to mutable objects in the environment; as well as 2) parameterized constraints on an agent's cognitive resources, e.g., limited capacity memory.

1. Solution to puzzle in Figure 1: A goes into 1 and C goes into 2.

3. We demonstrate the flexibility of VREE for representing agents having diverse strategies on two types of tasks, first on the classic block design task, which is a very standard test of human visuospatial ability (Kohs, 1920), and then on six subtests from the Leiter-R.

2. Proof-of-Concept Experiment for Sufficiency, Using the Leiter-R Test

The Leiter-R (Roid & Miller, 2011) is a test battery for evaluating nonverbal intelligence and related abilities in children and adults. It consists of 10 distinct subtests in a Visualization and Reasoning (VR) battery, and 10 more distinct subtests in an Attention and Memory (AM) battery, as shown in Table 1. Each subtest contains a series of multiple-choice-like problems of increasing difficulty; like the example in Figure **??** (similar to problems from the Leiter-R Visual Coding subtest), many problems involve several subproblems that must be answered from the same pool of answer choices. Responses are given, for the most part, by placing small cards representing the answer choices into slots representing the problem queries.

We hypothesized that most, if not all, items across all of the Leiter-R subtests could be solved by strategies that used only a small number of fundamental visuospatial operations, like computing visual similarity. What differs across subtests is how these fundamental operations are combined. In other words, it seemed reasonable that a fairly parsimonious domain-specific language for visuospatial reasoning could be sufficient for representing effective solutions across the variety of Leiter-R subtests, with approaches for each subtest differing only in their control knowledge, i.e., strategy.

2.1 Language and implementation details

In particular, a strategy S for reasoning through a subtest is some sequence of visual reasoning operations $(v_1, v_2, \ldots, v_n)$ and control operations $(c_1, c_2, \ldots, c_n)$. Table 1 describes each subtest as well as the strategy that we implemented to solve it.

Visual reasoning operations take images as inputs and process them to yield other images or numeric values. Control operations are similar to programming language constructs like loops and if statements that direct the flow of information between operations. When reasoning with a particular strategy, inputs from the subtest items can be processed up by any of the operations and the results generated could be passed on to other operations or stored for later use within the strategy. Our language included four different visual operations: similarity, scaling, rotation and containment.

Similarity. In almost every task strategy used by our models, the similarity operator is required. Formally, similarity s can be defined as $s(i_1, i_2) - > r[0, 1]$, where the inputs i_1 and i_2 are images, and the output r is a real number between 0 and 1. The metrics available in the system are the Jaccard similarity and the Euclidean distance.

The Jaccard similarity $s_J(A, B)$ between two images A and B is computed as the ratio of the size of the intersection of pixels in both images and the size of the union of pixels in both images:

$$s_J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

When using the Euclidean distance as a base for similarity, the two images to be compared are resized to be of the same size, and the euclidean distance $E(A, B)$ between the two is computed.

Table 1. All twenty tests in the Leiter-R Roid & Miller (2011), listed with brief descriptions of what the test requires and the strategies that were adopted in the model to solve the test. Three tests were not attempted: Picture Context, because it requires more extensive semantic reasoning than the other tests; Paper Folding, because it requires reasoning about transformations in 3D space; and Attention Divided, because it involves a different format of testing materials thatn the other tests.

Subtest	**Description**	**Strategy**
Associated Pairs	Match an item with its corresponding item, learned after an earlier training period.	Store all the image pairs displayed during the learning stage and find the one that best matches the requested images with similarity operations.
Attention Sustained	Find and mark as many instances of a given image as possible from a collage of similar images.	Using the similarity operator, mark the pixel locations where matches of the given image is found in the collage.
Classification	Match images with other images with which they share a semantic relationship. Example, socks and shoes.	Determine semantic relationships from a supplied database, and match images to make answer choices.
Design Analogies	A series of matrix based geometric analogy problems.	Determine trends in the matrix by finding a gradient over containments and similarities. Use the gradient to predict missing items in the matrix.
Form Completion	Match pieces broken apart to one of several whole images.	Choose piece as search target. Find regions in images with similar density, and then rotate piece in each region to find, and choose, best match.
Figure Ground	Find the image of an item or object from a larger image.	Find corners in the image and find the best matching corners in the response cards.
Forward Memory	Remember a sequence in which a series of images were progressively presented.	Store images as test progress and find similarities to stored images to provide the sequence.
Figure Rotation	Match an image with its rotated counterpart.	Fully rotate images in steps until a similarity metric meets a given threshold.
Immediate Recognition	Recognize items that have been removed from another collage displayed earlier.	Find corners in image and form triangles that are later matched to pinpoint exact locations.
Matching	Match items to their exact similar counterparts.	Match items with cards that have the highest similarity.
Picture Context	Select items that are semantically related to objects in a given image.	-
Paper Folding	Select images that are either folded or unfolded representations of other images presented.	-
Reverse Memory	Recollect items that were seen in the earlier Forward Memory task.	When solved after Forward Memory, use the stored image sequence and the same strategy used for Forward Memory.
Repeated Patterns	Find and complete a pattern in a sequence of items	Use a strategy similar to Design analogies in a single dimension instead of two.
Spatial Memory	Remember where the items are placed in a two dimensional space.	Store images as they are presented with their locations.
Sequential Order	Figure out a sequence in a series of images.	Use the same strategy for repeated patterns.
Visual Coding	Perform simple arithmetic with rules defined as images.	Store the relationships between images and match them to their similar counterparts during evaluation.
Delayed Recognition	Recognize items from the Immediate Recognition task without looking at the original images.	Run right after immediate recognition with all the “seen” images stored.
Delayed Pairs	Remember image counterparts from the Associated Pairs task.	Run right after associated pairs with all the “seen” images stored.
Attention Divided	Identify items from images while sorting another set of images.	-

With the distance computed, the euclidean similarity $s_E(A, B)$ can be computed as:

$$s_E(A, B) = \frac{1}{1 + E(A, B)}$$

Containment. The containment operation, provides a measure of the relative size difference between objects in two image patches. Formally, containment can be represented as $c(A, B, bg) -> r$ where A and B are images of the same dimension, and bg is the value of the background pixel. Containment can be computed as the ratio of all the foreground pixels (pixels that are not of the background color) in image A to those in image B.

Rotation and Scaling. Unlike the earlier operations described, the output of these two operations are images. Rotation results in an image that is rotated about its center, and scaling results in an image that has been resized. Formally, rotation r can be represented as $r(A)- > B$ where A is the input image and B is a rotated version of A. Likewise, scaling z can be represented as $z(A)- > B$ where A is the input image and B is the rotated version of A. Because these two do not yield any numerical results, they are mostly used for intermediary computations.

2.2 Leiter-R results: Sufficiency

Out of the twenty subtests on the Leiter-R test, we were able to produce implementations for all but the Picture Context, the Paper Folding, and the Attention Divided tasks. Of these 17 tasks, 8 were completely solved while the others were partially solved, as shown in Figure 2.

Most of the tasks that were fully solved were from the Attention and Memory (AM) battery of the test, which essentially requires just holding images in memory for later recognition or search. Hernández-Orallo et al. (2016) argues that it is "pointless to apply a memory test to a computer model," and while our results on these subtests may seem trivial, we expect that it is nevertheless fruitful to study such tasks especially within the broader ecosystem of visuospatial reasoning, and especially as we study agent performance under more realistic attention or memory constraints.

On the other hand, tests in the Visualization battery were a lot harder to develop successful strategies for, especially as the Leiter-R poses additional challenges as problems overtly or subtly change characteristics within the same subtest. Thus, a single strategy may only solve a few items in a given subtest.

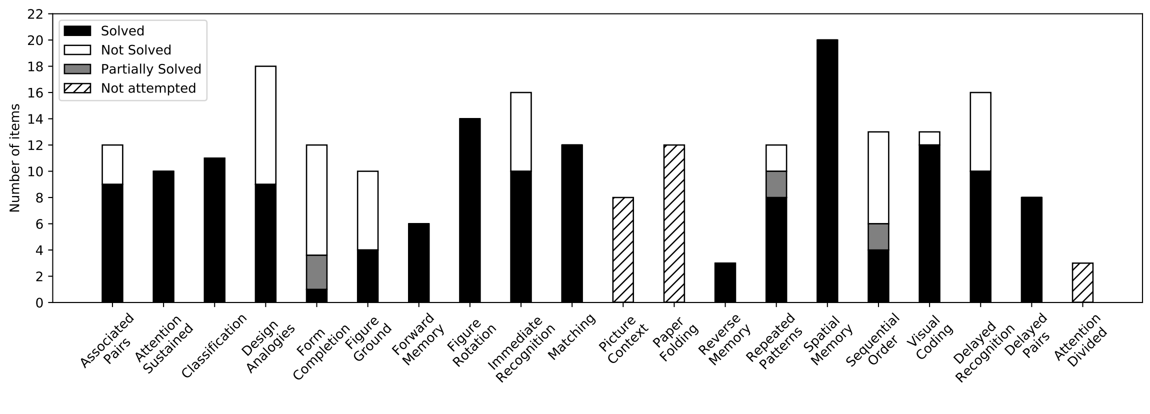

Figure 2. Results from proof-of-concept experiment to show sufficiency of visuospatial reasoning language for solving Leiter-R subtests.

It was interesting to observe how few operations were required to properly reason through the entire Leiter-R test. Of all the operations, the similarity operation seemed to be the one that affected results the most. The choice of the metric used for computing the similarity caused variations in the results that were obtained. For the results in Figure 2, all similarities were computed with the Jaccard Metric. Given the simplicity of the images in the Leiter-R, the Jaccard Metric works well, especially when the images have the same size and overlap properly. Due to the inherent imperfection in placing pages to be scanned, some images were slightly rotated. Thus, all images to be compared were automatically scaled such that they both had the same size. For images that were slightly rotated, however, a manual correction was performed.

3. The Visuospatial Reasoning Environment for Experimentation System

Next, we describe the design of an integrated environment for conducting computational experiments in the domain of visuospatial reasoning, to study strategy diversity (as described in this paper) as well as strategy selection, adaptation, and discovery (as described in Future Work).

The Visuospatial Reasoning Environment for Experimentation (VREE) is organized as an *environment* in which *objects* and *agents* interact. Agents in the environment contain a simplified cognitive architecture, which provides the resources for instantiating reasoning models. Since the primary focus of our work on VREE is the study of visual reasoning, the agents implemented perform much of their reasoning with knowledge that is represented as visual images.

For that reason, agents have visual attention mechanisms with which they "look" at items in the environment to acquire their inputs. And, when it comes to providing outputs, or extracting more information from the environment, agents can manipulate objects with *affordances* that exist in the environment. See Figure 3 for a block diagram of how the various parts of VREE work together.

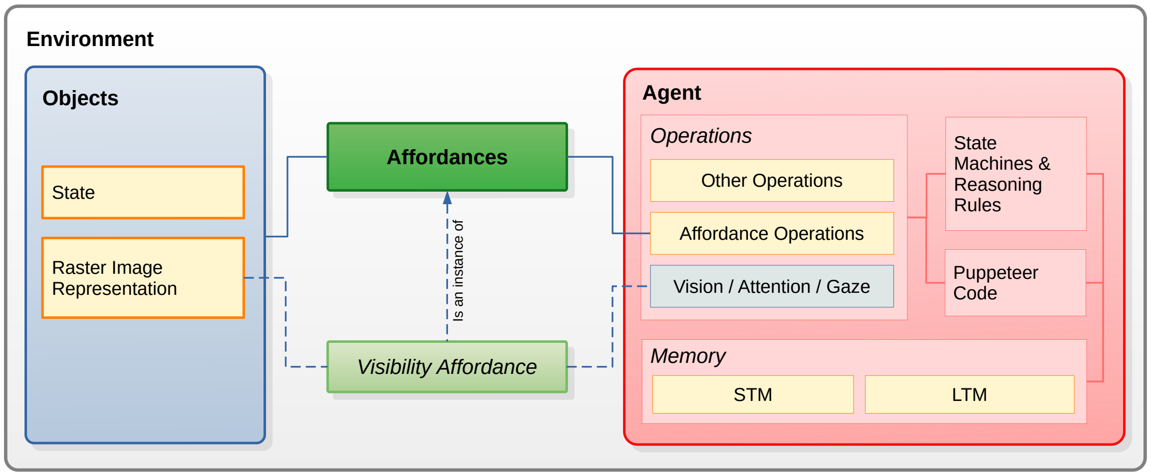

Figure 3. A system diagram of VREE that shows how the objects, affordances and agents relate in the environment. Multiple instances of objects and affordances, along with the built in visibility affordances interact with agent instances to reason through tasks.

3.1 The VREE Environment

The environment in VREE represents the world which needs to be reasoned about. All objects and agents reside directly in this environment. Formally, the environment can be represented as a tuple containing a set of objects, a set of affordances, a set of agents, and a TimeKeeper routine that coordinates the timing and activities of the agents.

$$Environment = \langle \textbf{Objects}, \textbf{Affordances}, \textbf{Agents}, TimeKeeper \rangle \quad (1)$$

The entire environment is modeled to exist on a two-dimensional plane, which is organized in a top-down fashion. This choice was made because, most of the tasks to be studies with VREE are intelligence tests that are usually administered in a table-top format. Two dimensional representations also simplify visual perception and are easier for computation. Ultimately, by adopting two dimensional views, the entire environment can be rendered as a raster image for visualization, and parts of it can be rendered for the agent's perception.

The environment also coordinates the interaction between these elements by providing the timing of their operations through the *TimeKeeper* routine. The time keeper essentially keeps a timer that is advanced in time-steps. In each time-step, the environment ensures that all necessary affordances are activated, and agents are given the opportunity to reason about the current state of the environment. See Algorithm 1 for a pseudocode representation of this algorithm.

Algorithm 1: $TimeKeeper$ routine from the Environment that coordinates the activites of agents, affordances and objects

```
do
    foreach $agent \in Environment.$***Agents*** do
        foreach $object \in Environment.$***Objects*** do
            foreach $affordance \in Environment.$***Affordances*** do
                if $affordance.AgentType = affordance.Type(agent) \wedge affordance.ObjectType = Type(object)$
                then
                    add $affordance.Actions$ to $agent.Operations$
        execute $agent$'s next step
forever;
```

3.2 Objects in VREE

Just like the environment, objects in VREE are also represented as two-dimensional top down views. Objects serve two primary purposes: convey information the agents take as input, and provide an avenue for the agents to communicate their results. Agents can get their inputs from objects by "looking" at them, and in cases where an agent needs more information, or needs to communicate an output, the agent can effect changes to the environment by manipulating objects.

Every object is an instance of an object type. The object type defines what the internal state of the object is. Objects also have a two dimensional coordinate location, and a raster image representation. Formally, an object can be represented as follows:

$$Object \Rightarrow \langle Type, State, Location, Image \rangle \tag{2}$$

The location represents the object's position in the environment's space, and the state holds other properties the object may have. How an object is perceived by any agent is determined by the contents of the raster image representation. This image representation can be static or dynamically generated. When all the images of objects in an environment are rendered with respect to their locations and sizes, a visualisation of the entire environment is created.

3.3 Affordances in VREE

Affordances determine how agents and objects interact with each other in the environment. In VREE, there are two types of affordances: ones that represent the abilities agents have with respect to particular object types (agent-object affordance), allowing agents to explicitly manipulate objects' states, and the ones that represent how objects of different types affect each another when they interact (object-object affordance). Although affordances are used by agents and objects, they exist in the environment, and their rules are enforced by the environment. The possibility of multiple object instances of a particular object type existing means, any objects that are added to the environment are automatically subject to any affordances that already exist for the object type.

Every affordance contains a definition of the conditions that must be met before the affordance is trigerred, and a list of operations that must be executed to perform the actions of the affordance. The triggering of affordances automatically occur in the environment for object-object affordances. With agent-object affordances, however, agents have the option of triggering any affordances whose conditions are met in a given time-step.

An affordance is formally defined as follows:

$$Affordance = \langle Conditions, Actions \rangle \tag{3}$$

The conditions required to trigger affordances are logical expressions evaluated on variables from the states of associated objects, and from the agent's internal state. For object-object affordances, rules only alter the object states, while object-agent affordances have the additional ability invoke the agent's operations and adjust the object's state too.

3.4 Agents

Agents in VREE contain a cognitive architecture for visuospatial reasoning. There is spatial memory, to help in tracking the locations of "interesting" items, and a salience map, that helps in directing gaze and attention to items in the environment. Additionally, agents posses long-term and short-term memories for storing the knowledge needed for reasoning, and a collection of reasoning operations that primarily manipulate images. When solving specific problems, agents can contain variables that can simulate other requirements for the task.

3.4.1 Operations in Agents

The building blocks of an agent's reasoning are its operations. Inside VREE, operations are specific actions agents can take in a timestep. The capabilities of an agent are therefore defined by its operations. As far as implementations go, operations are functions within an agent that take no arguments. Affordances that agents have are also translated transparently into operations, so agents can use these affordances without explicit knowledge of their existence. By default, every agent has operations for controlling its gaze and manipulating its memory. An agent can move its gaze arbitrarily, or to objects highlighted in its salience map. Anything in the short-term memory of the agent can also be committed. With the primary mode of knowledge representation being visual imagery, operations in VREE are mostly imagery based.

3.4.2 Short-term and Long-term memory

Memory in VREE is split up into long term memory and short term memory. The main difference between the two is permanence: long term memory is built into the agent, and by current design, cannot be changed once the agent is up and running, while short-term memory is acquired through reasoning, and changes freely at runtime. Both memories are internally modelled as symbol tables, where facts to be remembered are given a unique key so they can be retrieved in an instant. The representation of the actual fact to be "remembered", could be either symbolic or iconic. This memory modeling scheme may not be comparable with actual biological processes, especially when considering reconstructive nature of long-term memory in humans, but for the purposes of studying strategies for reasoning, this simplified abstraction works well.

3.4.3 Rules for Reasoning

There are two mechanisms by which the rules, which determine how and when operations are performed — and in effect represent the agent's model of reasoning — can be executed. One approach involves controlling the agent with an external program — much like how a puppeteer controls a puppet, and the second approach allows agents to be more "autonomous," so they can reason with internal rules that are represented as state machines. When reasoning with external programs, the program decides exactly which operations would be executed. For reasoning with a state machine, however, an operation, or a group of operations, or even another entire state machine can be executed in any state. Transitions between these states are determined by conditions that are stored in a transition table within the agent.

Reasoning with an external program is more favorable when the task to be performed is complex. In comparison to state machines, the flexibility provided by the control structures in high level programming languages, like python, make it easier to implement complex reasoning models. In contrast, reasoning through state machines may not be as complex, but agents have the ability to modify their own transition tables, making it possible for agents to learn new rules.

3.4.4 Vision, Gaze and Attention

Agents in VREE acquire much of their information through a visual attention mechanism. This mechanism has a gaze window for collecting information from the environment, and a salience map for determining where to move this gaze window for attention. The entire mechanism runs on visibility affordances that are built into the environment. Through these affordances, an agent is able to maintain a salience map of the entire environment. This salience map, which can be considered in some ways to be the agent's peripheral vision, provides a view of objects existing in the environment without the fine details.

Two visibility affordances in the environment are responsible for managing the vision and gaze system. These affordances are the salience map affordance and the gaze affordance. The salience map affordance updates the salience maps of agents after every timestep, and the gaze affordance can be used by the agent to investigate items on its salience map.

There are no conditions for the salience map affordance so it always runs, and it is enforced to execute last after all other affordances and operations. Once all operations and affordances have executed, the salience map affordance updates agents' salience maps with the locations and size of every object. The map is stored as a name-coordinate pair, where each object is given a unique identifier (name) to which its corresponding location and size are mapped.

For each object already on the salience map, a gaze affordance is created that is active whenever the object is not completely obscured by another. The action offered by this gaze affordance is translated into an operation in the agent that moves the gaze window to cover the entire object.

Some objects may contain multiple areas of interest, such as a surface with different images or the different parts of a puzzle. Such objects can supply additional salience maps (mapped relative to their surface) that highlight regions of interest so an agent can pay attention. Internally, the salience map affordance merges all salience maps (for both objects and parts of objects) into a single map, making it possible for attention to be distributed uniformly between objects and parts of objects.

4. Demonstrating VREE with the Leiter-R

We revisited six subtests from the Leiter-R to implement reasoning agents within the new VREE environment. In comparison to our earlier experiments, VREE agents must be much more carefully specified in terms of the interactions among various components of their architecture, including how they interact with the external environment, as described above.

For these Leiter-R subtests, the VREE environment we created had two object types. We had the *easel* object type for easels, and the card object type for response cards. With each subtest, we took actual scanned images from the original test material and used those as the image representations. To allow agents to move the cards to their selected slots, we added two affordances — one to move cards in, and another to move them out.

All the scanned images presented as inputs in the environment were manually annotated with the regions the agent should pay attention to. These annotated regions were selected from the distinct images of items from the easel slots and the images on the cards.

For all of the subtests, we implemented two high-level strategies within VREE. In the first strategy, the agent begins with a main image from one of the cards (i.e. answer choices), and performs whatever search or reasoning is necessary to match this card with one of the problem slots (i.e. the individual subproblem queries). We call this the card-to-slot strategy. In the second strategy, the agent begins with a main image from the easel (i.e. the subproblem queries) and tries to connect this particular query with a specific card (i.e. answer choice). We call this the slot-to-card strategy. While these strategies may seem equivalent, in fact many items on the Leiter-R seem to be more or less difficult to solve using one versus the other of these strategies (even, sometimes, for different subproblems within the same problem). Part of the reason for this is that, because there are a limited number of cards, every subproblem that gets solved can make it easier to solve the next subproblem (i.e. by reducing the number of remaining cards to choose from), or it might make solving the next subproblem impossible (if an incorrect card was chosen and used too early).

We also simulated forgetfulness in the agents by adding random noise to the short term memory after every time-step. As time-steps pass, random noise builds up, corrupting items in the short term memory become corrupted. This feature can be turned off, and the rate at which noise is added can also be adjusted. The memory was set up to fade at a rate of 10% every timestep.

While running our tests, we measured the accuracy of each agent having a given strategy and either with or without forgetfulness. Results are presented in Figure 4. As this figure shows, even the small set of strategy variations we implemented here shows considerable effects on performance. In addition, for each agent, we additionally have traces of the agents gaze and manual actions while solving each test item, though these observations are not shown here.

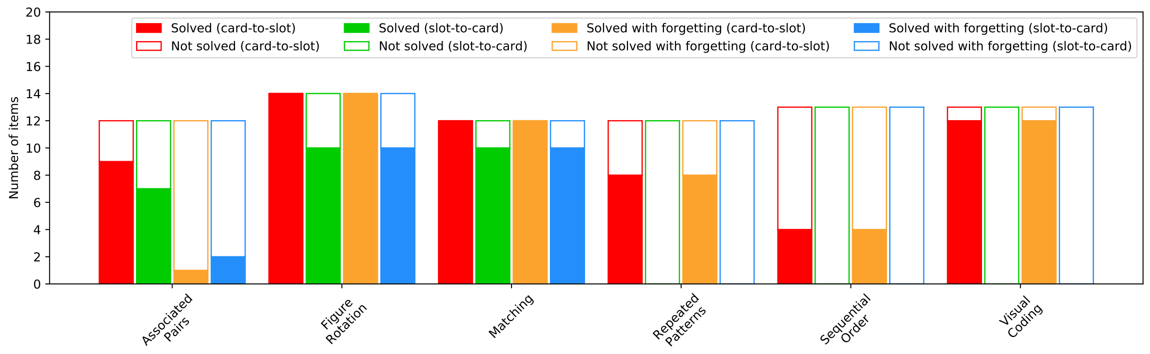

Figure 4. A plot of the number of items successfully completed by various VREE agents when reasoning through six different Leiter-R subtests.

4.1 Leiter-R Walkthrough for the Visual Coding Task

The task displayed in Figure 1 is an example of an item from the Visual Coding task in the Leiter-R. When implemented in VREE, the environment for this item is set up as follows:

1. Each of the images labeled A through F in Figure 1 (those in the gray box) are represented as individual *card* objects.
2. The rest of the image (the part without the gray box) is placed on the easel object. For this particular test item, the manual annotations that drive attention on the easel image are placed on the individual icons.
3. For the images in the top row that represent the rules, annotations are made to explicitly lay out the mappings. When it comes to those on the bottom row, the annotations expose

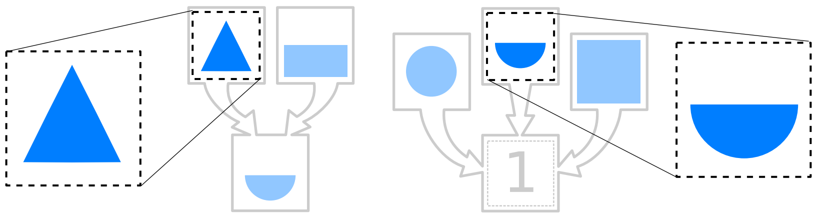

Figure 5. A diagram showing how annotations are extracted from the problem for reasoning on the Leiter-R task.

the sequence of images that represent a problem. See Figure 5 for how the annotations are extracted.

When solving for the box labeled 1 in the problem from Figure 1, the model first looks at the three image clues. Next, a search is performed on all the rules, from the top row, for any pair that are found in the trio of images in the problem — a circle, a semi-circle and a square. In the case of this particular problem, the first match will be in the first rule image from the right. This rule maps the circle and semi-circle to a triangle.

Moving on, the triangle replaces the circle and semi-circle, limiting the search to be for a square and a triangle. In the second round of searches, a match for the square and triangle are found in the middle of the top row. This new match reduces the images being searched for to a single inverse semi-circle. When the search is just for a single image, the answer cards (labeled A through F) are searched. A match is found in option A, thereby causing the agent to select that card as its answer.

5. Demonstrating VREE with the Block Design Test

The block design task is a visual reasoning task that was first introduced by Kohs (1920). It was designed to provide a means to measure non-verbal reasoning skills, and it is usually included as part of standardized IQ test batteries such as the Wechsler Adult Intelligence Scale (WAIS) (Wechsler, 2008). When solving the block design test, a test taker is presented with a group of blocks placed in a block bank, and a design to be replicated. The test taker is supposed to replicate this design by placing each block down in a designated construction area with the block's upward face displaying a fragment of the given design image.

Note that while the current work is conceptually related to earlier computational research on the block design test by members of our research team (Kunda et al., 2016), the current VREE implementation is quite different from our earlier work, with the model having a very different structure. In particular, our earlier block design model was task-specific, whereas VREE has been designed to provide a generalized environment and agent archtitecture for solving a wide variety of visuospatial reasoning tasks. Thus, objects, affordances, and possible actions taken by the model are all defined in a more generalized form in the current VREE implementation. The current work also explores many more strategies, and provides a different portrayal of behavioral measures, than were explored in our earlier work.

5.1 Representing the Block Design Task in the VREE System

The tactile nature of the block design task makes it a great candidate for investigation in the VREE system. In translating the physical block design into the VREE system, we defined two different object types. These were the *Block* object type, used to represent the physical blocks, and the *Design* object type, used to represent the design. For the experiments in this work, there was always a single agent in the environment at any time, and this agent's role was to solve the tasks. Individual components of VREE were set up as follows for the Block Design Task:

1. The internal state of any block *Block* was the name of its current face.
2. The solving agent was given a simulated hand that could pick and hold blocks. This hand was implemented through a state variable in the agent, $HandContents$, which could either be empty or contain a reference to a block.
3. The design was pre-segmented into its individual cells, and was represented by a $Design$ object whose internal state held the size, orientation (diagonal or straight), and the individual faces for each cell. Because each cell had to be attended individually, the locations and sizes of each individual cell was presented as part of the salience map of the design. We chose to pre-segment the design because the aim of our work was not to measure perception, but to investigate different goal solving strategies.
4. The long-term memory of the agent was populated with coordinate information about two regions in which blocks could be placed: the block bank, where new blocks could be picked, and the construction area, where blocks could be assembled. The construction region was further separated into specific cell locations to help the agent in assembling the blocks.
5. The long-term memory also contained a symbolic model of the block which contained information about the different faces and the transitions between them.
6. There were ten different affordances, split into two groups that allowed the agent to pick and spin blocks, and move them around the different regions of the environment.

The reasoning model for the block design task is built around a general template strategy. The flow of reasoning for this strategy leaves out certain key steps so they could be implemented by other sub strategies. Reasoning proceeds by considering each cell in the design as a smaller intermediate

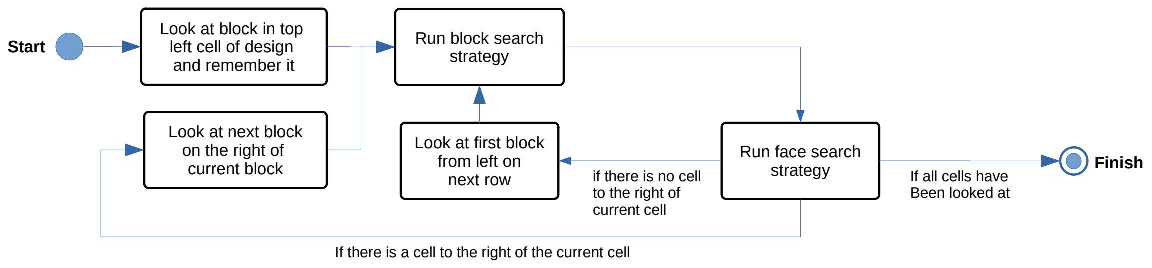
Start
Look at block in top left cell of design and remember it
Run block search strategy
Look at next block on the right of current block
Look at first block from left on next row
Run face search strategy
if there is no cell to the right of current cell
If all cells have Been looked at
Finish
If there is a cell to the right of the current cell

Figure 6. State machine representation of the general template strategy. Each rectangle represents a state node that has an activity to be performed once that state is reached. The edges between the nodes represent the conditions that must be met before a transition can occur on the edge. Edges without transitions conditions are considered to be always true.

goal for which a corresponding block should be found. For each immediate goal, the agent looks at the block for the goal and memorizes it the short-term memory. With the block in memory, the first sub-strategy runs to find the best block to pick from the block bank. After a candidate block is found, a second sub-strategy is run to find the right face for the block to be placed in the corresponding part of the design. The matching face that is found is placed in the construction area to signify completion of the cell's subgoal. This process is repeated until the entire design is completed. The state machine used for implementing this strategy is shown in Figure 6.

When it comes to the first sub-strategy for picking the block from the bank, there are two possible approaches. For the first approach, the agent picks a block from the bank and works with it no matter its face. With the second possible strategy, the agent scans the blocks in the bank so it could pick the first block that had the same composition of colors as the image in memory.

The next sub-strategy which runs after a block is found, is for searching for the face on the block that best matched the face in memory. There are three different strategies for this phase. The simplest strategy for finding the face is to spin and flip the block randomly until the right face is found. The second best strategy works by detecting the colors of the upward face and further flipping the block depending on the colors detected after each flip. A final strategy that does not rely on any form of imagery runs a regular depth-first search on the block faces, using a symbolic model of the block and transitions between faces stored in the agent's long-term memory.

Algorithm 2: Imagery search algorithm for finding the matching face to the cell in memory.

```
$ActionQueue \leftarrow \{Nothing, FlipUp, FlipRight\}$;
$BlockSpun \leftarrow False$
while $ActionQueue$ is not empty do
    pop $nextAction$ from $ActionQueue$;
    perform $nextAction$;
    if face on block matches with cell in memory then
        end loop;
    if current face has red and white $\wedge \neg BlockSpun$ then
        $ActionQueue \leftarrow \{SpinClockwise, SpinkClockwise, SpinClockwise,$
        $FlipUp, FlipRight, FlipRight, FlipRight\}$;
        $BlockSpin \leftarrow True$
```

5.2 Block Design Walk-through

To fully demonstrate how the agent reasons through a problem, here is a walk-through of the steps taken to solve the puzzle presented in Figure 7. For this walk-through, the agent is working with the sequential strategy for picking the next block, and the imagery strategy for finding the face that matches that of the cell to be completed.

First, the agent places its gaze window on the image in the top left cell of the design as it starts to work its way one cell at a time (see Figure 7-1). According to the sequential strategy, the agent

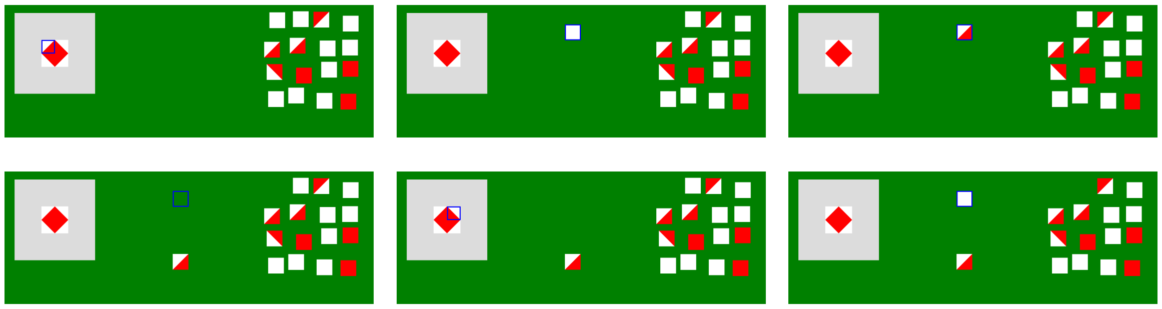

Figure 7. A view of the VREE environment for a few initial steps while solving a block design item. The design to be replicated is on the left, the block bank is on the right, and the final construction is in the center. The blue square indicates the current gaze position of the VREE agent.

picks the next available block from the top-left corner of the block bank to work with (see Figure 7-2). In the case of this particular run, this block happens to have a white face. The agent attempts a match to find out if this face matches that of the cell. This fails and causes the agent to flip the block up. A red and white face in the SE orientation is exposed (see Figure 7-3). When this face is compared with the current cell, a match is found, and the block is moved into the construction zone,and the process continues.

5.3 Results from the Block Design Task

We evaluated the block design task with seven designs, and six strategies. The six strategies were obtained by filling in the sub-strategy stages of the general template strategy. On the first sub-strategy stage stage, there was an option from two different sub-strategies, and for the second stage there was an option from three. The two strategies for the first stage were the *closest looking* search and the *sequential search*, while the three for the second stage were the *depth first* search, *imagery* search, and *random flips*. These combinations yielded the following distinct strategies: *closest looking depth*, *closest looking imagery*, *closest looking random*, *sequential depth*, *sequential imagery*, and *sequential random*. While running these strategies, we recorded the total number of steps the models took as a measure of their response times. The values obtained for the different individual designs are shown in Figure 8.

Our second experiment hinged on the simulation of forgetfulness in VREE. To demonstrate the effect of this feature, we re-run all the experiments in the block design section with the memory fading at a rate of 10% every timestep. This means the agent will totally forget anything in short-term memory after 10 timesteps. The results for this second run are shown in Figure 9.

The two plots from the experiments on the block design task show how the six different strategies performed under different conditions. It can be seen from the charts that the strategies that relied on imagery and randomness performed worse than those that took a more symbolic approach. This is even more amplified in the case were the agent's working memory was continually corrupted. Agents relying on randomness even did much worse in this case. Regardless of the number of steps taken, all strategies ended up solving the block design correctly.

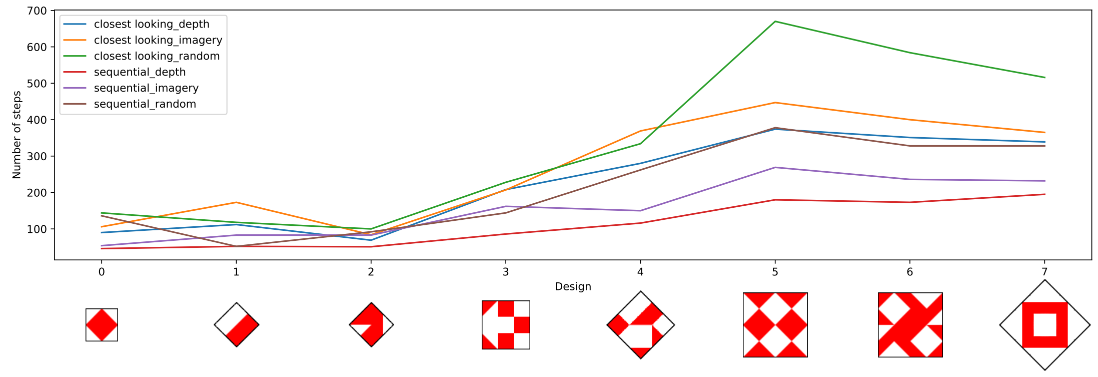

Figure 8. A plot of the total number of steps taken by the agent when reasoning with the six different strategies. The solid line represents the average over 10 different runs, and the transparent section represents the range of values within one standard deviation.

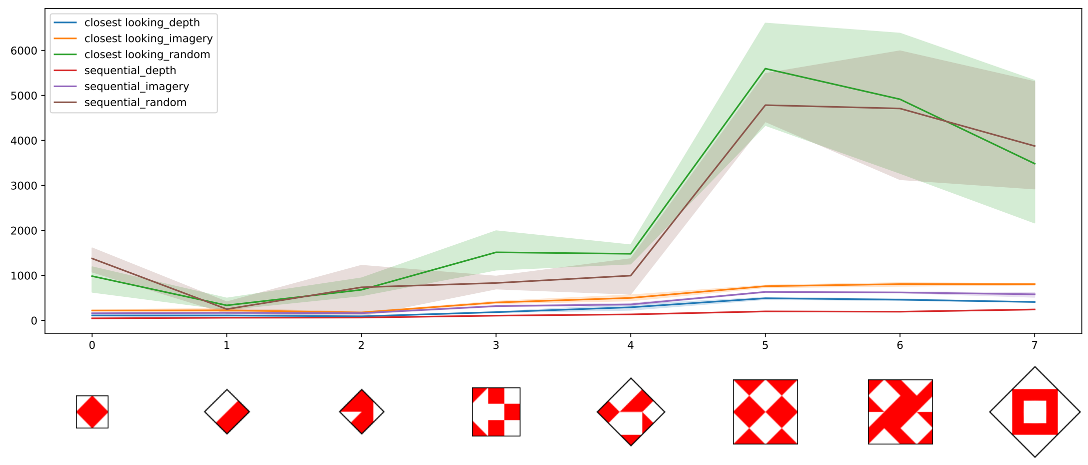

Figure 9. A plot of the total number of steps taken by the agent when reasoning with the six different strategies. For this plot, the agent's forgetfulness has been turned on. The solid line represents the average over 10 different runs, and the transparent section represents the range of values within one standard deviation.
.

6. Related Work

At their core, most major cognitive architectures depend largely on symbolic information for knowledge representation. Because a large number of these systems are based on production systems, all the knowledge needed for reasoning, including the rules, are stored in a homogeneous symbolic/propositional format. Some spatial reasoning tasks, however, can also be solved using imagery representations, which has led to the creation of extensions (Lathrop & Laird, 2007; Wintermute, 2012) to some of these traditional architectures that provide spatial reasoning, allowing models built with these architectures to employ multi-modal reasoning about visuospatial tasks . But, even so, these extensions at some point still transform their "perceptual" inputs into the symbolic formats needed by their host architectures for reasoning.

There are also other cognitive systems that were built from the ground up for visuospatial reasoning. Phaeco (Foundalis, 2006) is a system that was built to solve Bongard Problems. It was organized as a cognitive architecture that reasoned in part with imagery operations, pattern matching, and learning mechanisms. Another dedicated architecture that extracts spatial information from images using the concept of sketch understanding is CogSketch (Forbus et al., 2011). Cogsketch has been succesfully applied to analogy problems like the Ravens Progressive Matrices (Lovett et al., 2007; Lovett & Forbus, 2011).

6.1 Reasoning with Images

Kunda et al. (2013) implemented the Affine-and-Set Transformation Induction (ASTI) model that reasons through matrix geometric analogy problems using only image representations and operations. Specifically this system was intended to solve the Ravens Progressive Matrices. It worked by trying out different operations across the matrix until it figures out a series of operations that creates a change in a row or a column. Once this operation is found, animage is synthesized with this operation and compared with the possible answers to find a match. When tested on the standard Ravens Matrices, ASTI was able to score 50 out of the 60 problems without extracting any form of verbal or propositional information (Kunda, 2013).

Similar work has also been done with the block design task (Bringsjord & Schimanski, 2003; Kunda et al., 2016), the punched hole paper folding task (Ainooson & Kunda, 2017), and the Figure Ground task from the Leiter-R (Palmer & Kunda, 2018).

The paper folding task is a pencil and paper task where the participant is supposed to imagine how a piece of paper looks unfolded, after it had been earlier folded in a particular pattern, and had a hole punched in it. The model for solving the paper folding task constructed a pseudo three-dimensional representation of the folded paper using a stack of two dimensional images (Ainooson & Kunda, 2017).

Bringsjord & Schimanski (2003) solved the block design task using actual physical blocks and a robot named PERI. PERI was able to receive visual information about the state of the blocks, and with image matching operations, it was able to solve the task by moving the actual blocks into place. Kunda et al. (2016) used a similar approach for the block design solver, except everything was solved in a simulated environment, with an agent that could simulate properties such as a limited-capacity visual short-term memory.

7. Discussion and Future Work

A wide variety of computational models have been developed that do very well on various intelligence tests, including tests similar to the visuospatial reasoning tasks studied here (Hernández-Orallo et al., 2016). Some models are purpose-built to work on a single test (e.g. single test sufficiency), while others aim to work through multiple tests (e.g. multiple test sufficiency). Several also look at strategy variations, ranging from simple differences in feed-forward mechanisms (Kunda et al., 2013) to much more complex, iterative feedback loops of reasoning and re-representation (Lovett & Forbus, 2017). A very small number of models actually aim to come up with strategies themselves, falling within inductive programming paradigms (Schmid & Kitzelmann, 2011). And more recently, increased attention is being given to the problem of how computational models might take the tests just as humans do, i.e. through receiving task instructions, either verbally or through demonstrations (Laird et al., 2017; Lázaro-Gredilla et al., 2019; Kunda, 2019; Chollet, 2019), instead of being manually programmed with task definitions up front.

Future work on VREE is going to be centered around the incorporation of learning mechanisms into the definition of operations, the knowledge of tasks, and the formulation of strategies for solving tasks. Initial priority will go into learning the formulation of strategies, through program induction techniques, with inspiration from Lázaro-Gredilla et al. (2019), Johnson et al. (2017), and Balog et al. (2017). This line of work will be key in in understanding strategy diversity, i.e., how individual intelligent agents form their strategies based on their own cognitive resources, preferences, and reasoning and learning capabilities.

Acknowledgements

We thank Mohamed El-Banani, whose earlier block design modeling research identified some of the key conceptual motivations for this work, and the anonymous reviewers for their helpful feedback. This work was supported in part by NSF BCS Award #1730044, NSF OIA Award #1936970, and NSF DGE Award #1922697 through the Neurodiversity Inspired Science and Engineering (NISE) NSF Research Traineeship (NRT) program.

References

Ainooson, J., & Kunda, M. (2017). A computational model for reasoning about the paper folding task using visual mental images. *Proceedings of the 39th Annual Conference of the Cognitive Science Society, London, UK*.

Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., & Tarlow, D. (2017). DeepCoder: Learning to Write Programs. *arXiv:1611.01989 [cs].*

Bringsjord, S., & Schimanski, B. (2003). What is artificial intelligence? psychometric AI as an answer. *Proceedings of the 18th international joint conference on Artificial intelligence* (pp. 887–893). Acapulco, Mexico: Morgan Kaufmann Publishers Inc.

Chollet, F. (2019). On the measure of intelligence. *arXiv preprint arXiv:1911.01547 [cs.AI].*

Forbus, K., Usher, J., Lovett, A., Lockwood, K., & Wetzel, J. (2011). CogSketch: Sketch Understanding for Cognitive Science Research and for Education: Topics in Cognitive Science. *Topics in Cognitive Science*, *3*, 648–666. From `http://doi.wiley.com/10.1111/j.1756-8765.2011.01149.x`.

Foundalis, H. (2006). Phaeaco: A cognitive architecture inspired by bongard's problems [ph. d. thesis]. *Indiana University, Indiana, Bloomington.*

Hernández-Orallo, J., Martínez-Plumed, F., Schmid, U., Siebers, M., & Dowe, D. L. (2016). Computer models solving intelligence test problems: Progress and implications. *Artificial Intelligence*, *230*, 74–107.

Johnson, J., Hariharan, B., Van Der Maaten, L., Hoffman, J., Fei-Fei, L., Zitnick, C. L., & Girshick, R. (2017). Inferring and Executing Programs for Visual Reasoning. *2017 IEEE International Conference on Computer Vision (ICCV)* (pp. 3008–3017). Venice: IEEE.

Kohs, S. C. (1920). The block-design tests. *Journal of Experimental Psychology*, *3*, 357.

Kunda, M. (2013). *Visual problem solving in autism, psychometrics, and ai: the case of the raven's progressive matrices intelligence test*. Doctoral dissertation, Georgia Institute of Technology.

Kunda, M. (2019). Nonverbal task learning. *Seventh Annual Conference on Advances in Cognitive Systems*.

Kunda, M., El Banani, M., & Rehg, J. M. (2016). A computational exploration of problem-solving strategies and gaze behaviors on the block design task.

Kunda, M., McGreggor, K., & Goel, A. K. (2013). A computational model for solving problems from the raven's progressive matrices intelligence test using iconic visual representations. *Cognitive Systems Research*, *22*, 47–66.

Laird, J. E., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Land, M., & Tatler, B. (2009). *Looking and acting: vision and eye movements in natural behaviour*. Oxford University Press.

Lathrop, S. D., & Laird, J. E. (2007). Towards incorporating visual imagery into a cognitive architecture. *Proceedings of the eighth international conference on cognitive modeling* (p. 25). From `http://iccm-conference.org/2007/files/lathrop___laird.pdf`.

Lázaro-Gredilla, M., Lin, D., Guntupalli, J. S., & George, D. (2019). Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs. *Science Robotics*, *4*, eaav3150.

Lovett, A., & Forbus, K. (2011). Cultural commonalities and differences in spatial problem-solving: A computational analysis. *Cognition*, *121*, 281 – 287. From `http://www.sciencedirect.com/science/article/pii/S0010027711001727`.

Lovett, A., & Forbus, K. (2017). Modeling visual problem solving as analogical reasoning. *Psychological review*, *124*, 60.

Lovett, A., Forbus, K., & Usher, J. (2007). Analogy with qualitative spatial representations can simulate solving raven's progressive matrices. *Proceedings of the Annual Meeting of the Cognitive Science Society*.

Palmer, J. H., & Kunda, M. (2018). Thinking in polar pictures: Using rotation-friendly mental images to solve leiter-r form completion. *Proceedings of AAAI*.

Roid, G. H., & Miller, L. J. (2011). Leiter international performance scale-revised (leiter-r). *Madrid: Psymtec*.

Schmid, U., & Kitzelmann, E. (2011). Inductive rule learning on the knowledge level. *Cognitive Systems Research*, *12*, 237–248.

Tversky, B. (2005). Visuospatial reasoning. *The Cambridge handbook of thinking and reasoning*, (pp. 209–240).

Wechsler, D. (2008). Wechsler adult intelligence scale–fourth edition.

Wintermute, S. (2012). Imagery in cognitive architecture: Representation and control at multiple levels of abstraction. *Cognitive Systems Research*, *19*, 1–29.