NORTHWESTERN UNIVERSITY

Spatial Routines for Sketches: A Framework for Modeling Spatial Problem-Solving

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Andrew Lovett

EVANSTON, ILLINOIS

June 2012

ABSTRACT

Spatial Routines for Sketches: A Framework for Modeling Spatial Problem-Solving

Andrew Lovett

Spatial problem-solving tasks are often used to evaluate people's cognitive abilities. For example, Raven's Progressive Matrices is a popular intelligence test. In it, an individual is shown an array of two-dimensional images, with one image missing. The individual must compare the images and identify a pattern of differences between them, in order to solve for the missing image. Performance on tasks such as Raven's and geometric analogy ("A is to B as C is to..?") correlates strongly with performance on many other ability tasks, in the spatial, verbal, and mathematical domains. Thus, these tasks appear to depend on core, general-purpose representations and processes. However, it is as yet unclear what those representations and processes are.

To better understand these tasks, we developed Spatial Routines for Sketches (SRS), a general framework for modeling spatial problem-solving. SRS is based on a set of psychological claims about how people perform spatial problem-solving: 1) When possible, people use qualitative representations describing features such as relative position or orientation, rather than exact numerical values. 2) Spatial representations are hierarchical. A given image might be represented as object groups, individual objects, or the parts within each object. 3) Qualitative spatial representations can be compared via structure-mapping. Structure-mapping involves aligning the relational structure in two representations to find the corresponding elements.

Three task models were built within the SRS framework: geometric analogy, Raven's Progressive Matrices, and the oddity task, in which one sees a set of images and picks the one that is different. The three task models use identical representations and similar processes. Thus, they allow us to test the generality of the psychological claims, as well as the representations and processes that implement these claims.

Each task model was compared against human performance. All three models perform at least as well as human adults, and problems that are difficult for the models are also difficult for people. This supports the claims on which the models are based. Furthermore, by ablating a model's ability to perform certain operations and examining the error pattern this produces, we are able to produce new hypotheses about human reasoning.

ACKNOWLEDGEMENTS

This research was supported by NSF grant SBE-1041707, the Spatial Intelligence and Learning Center.

I'd like to thank my advisor, Ken Forbus. I felt very fortunate to be able to pursue exactly the research I wanted to do. This thesis has been a long journey, but Ken was a guiding light, and a constant source of support, even when I wandered far from the Computer Science path.

Thank you to Dedre Gentner for convincing me to come to Northwestern in the first place, and for helping me pretend to be a psychologist. Who knows, maybe I'll fool someone someday. Along with Dedre, I've enjoyed fruitful collaborations with Steve Franconeri, Justine Cassell, and Kristina Striegnitz. It was always fun branching out into new areas, and I learned useful lessons from each of them. Thank you also to Ian Horswill. While we unfortunately never collaborated, Ian provided an important voice on my thesis committee.

My seniors in the Qualitative Reasoning Group played a major role in my development. Praveen Paritosh, our dysfunctional papa, was immensely valuable for sounding out new ideas. Emmett Tomai bequeathed geometric analogy (and, by extension, my entire thesis) to me. Kate Lockwood understood the system like nobody else. Matt Klenk helped me move out of my comfort zone. Morteza Dehghani was technically my junior, but you'd never know it. He was a valuable collaborator and friend.

Madeline Usher saved my Science more times than I could count. In addition, Madeline was a terrific friend, always available for discussing work or non-work issues.

A big thank you to my other Evanston friends: David, Laura, Dan, Katie, and Noel. They kept things interesting throughout the years. Some of us have left, and some may be leaving soon (e.g., me, if I ever get a job), but this was a special time for me, and I hope for all of them.

Finally, I thank my parents, David and Ellen. You truly define unconditional love. I know you'll support me no matter what, even when (I mean, *if*) I abandon academia to make video games.

TABLE OF CONTENTS

1. Introduction 16
1.1 Motivation 19
1.2 Background 21
1.3 Claims 29
1.4 Contributions 39
1.5 Evaluation 40
1.6 Outline of the Thesis 41
2. Background 42
2.1 Hybrid Representations 43
2.2 Hierarchical Representations 55
2.3 Structural Comparison 61
2.4 Mental Rotation: An Example Domain 62
2.3 Strategic Comparison 68
3. Modeling Perception and Comparison 72
3.1 Existing Models 73
3.2 Perceptual Sketchpad 77
3.3 Image Comparison 91
3.4 Shape Comparison 97
3.5 Perceptual Reorganization 109
4. Spatial Routines for Sketches 112
4.1 Operation Categories 114
4.2 Spatial Operations 118
4.3 Other Operations 153

4.4 Spatial Routine Language 154

4.5 Gathering Data 166

5. Simulations 169

5.1 Representation 170

5.2 Task Comparison 173

5.3 Parameters and Sensitivity Analysis 174

5.4 Analyses 175

6. Geometric Analogy 177

6.1 Background 178

6.2 Model 181

6.3 Model Predictions 189

6.4 Behavioral Experiment 190

6.5 Simulation 200

6.6 Related Work 205

6.7 Conclusion 207

7. Raven's Progressive Matrices 209

7.1 Background 210

7.2 Model 215

7.3 Behavioral Experiment 225

7.4 Simulation 228

7.5 Discussion 235

7.6 Related Work 238

8. The Oddity Task 241

8.1 Background 242

8.2 Model 243

8.3 Simulation 247

8.4 Discussion 254

9. Predictions........ 259

9.1 Visual Comparison 259

9.2 Spatial Problem-Solving........ 263

10. Conclusions........ 267

10.1 Thesis Claims 267

10.2 Related Work........ 271

10.3 Future Work 273

References........ 279

Appendix........ 289

Appendix A: Modifications to the Structure-Mapping Engine 289

Appendix B: The Geometric Analogy Routine 292

Appendix C: The Raven’s Progressive Matrices Routine 298

Appendix D: The Oddity Task Routine........ 308

LIST OF FIGURES

Figure 1.1: Geometric analogy problem 16

Figure 1.2: Three spatial problem-solving tasks 18

Figure 1.3: An oddity task problem involving groups of objects 32

Figure 1.4: Two related geometric analogy problems, along with average reaction times and overall accuracies for human participants 35

Figure 2.1: Two oddity task problems from Dehaene et al. (2006). Pick the image that doesn’t belong ... 42

Figure 2.2: Stimuli from (A) Navon (1977, Experiment 4), and (B) Love, Rouder, and Wisniewski (1999) (B) 58

Figure 2.3: Two images that one might compare 60

Figure 2.4: Mental rotation stimuli from (A) Shepard and Metzler (1971), and (B) Cooper and Shepard (1973) 62

Figure 2.5: Reaction time results from Biederman and Bar (1999, Experiment 2) 67

Figure 3.1: Example stimuli from Love, Rouder, and Wisniewski (1999) 72

Figure 3.2: The CogSketch sketch understanding system 74

Figure 3.3: Sketch of a house 78

Figure 3.4: Examples of texture patches 79

Figure 3.5: Examples of space objects 79

Figure 3.6: Grouping examples 80

Figure 3.7: Examples of parallel edges, parallel objects, and parallel groups 82

Figure 3.8: Example representation with all expressions about Edge-4 in a parallelogram 85

Figure 3.9: A rotation between two triangles 85

Figure 3.10: Two images that might be compared, with the objects labeled 92

Figure 3.11: Two ambiguous mappings 94

Figure 3.12: Object pairs whose textures would be considered identical 95

Figure 3.13: Shape change examples 96

Figure 3.14: Shape rotation example 98

Figure 3.15: A) Spatial reflections. B) Shape reflections 101

Figure 3.16: Examples of lengthening 104

Figure 3.17: Example of part lengthening 104

Figure 3.18: Examples of part addition 105

Figure 3.19: Example of a subshape deformation 105

Figure 3.20: Corresponding edges for a symmetry mapping 106

Figure 3.21: Example of a scaled group 107

Figure 3.22: Complex reorganization facilitates this comparison 111

Figure 4.1: An ambiguous image 113

Figure 4.2: A: Oddity task problem from (Dahaene et al., 2006). B: Raven’s Matrix-type problem 115

Figure 4.3: Geometric analogy problems 117

Figure 4.4: A geometric analogy problem, as it looks in CogSketch 120

Figure 4.5: A Raven’s Progressive Matrices ‘A’ problem (not from the actual test) 120

Figure 4.6: A row of images for difference-finding 129

Figure 4.7: Raven’s Progressive Matrix problems requiring a literal pattern of variance 131

Figure 4.8: A row of images with first-to-last matches 135

Figure 4.9: Oddity task problems with orientation differences 136

Figure 4.10: A: A geometric analogy problem. B: The patterns of variance 137

Figure 4.11: Examples of *Infer-Shape* with reflections 140

Figure 4.12: Examples of *Infer-Shape* with deformations 141

Figure 4.13: Examples of *Infer-Shapes* with texture transformations 141

Figure 4.14: A geometric analogy problem 142

Figure 4.15: A geometric analogy problem with an added object 143

Figure 4.16: Four 3x3 Raven's Matrix problems 145

Figure 4.17: A geometric analogy problem with an unclear rotation 148

Figure 4.18: A Raven's Matrix problem requiring complex perceptual reorganization 149

Figure 4.19: A Raven's Matrix problem requiring texture detection 150

Figure 4.20: A Bongard problem from (Bongard, 1970) 163

Figure 4.21: The Routine Inspector on a geometric analogy problem 167

Figure 5.1: A: Geometric analogy problem (Evans, 1968). B: Raven's Matrix problem. C: Oddity task problem (Dehaene et al., 2006) 169

Figure 5.2: Each internal edge of a textured object was added as a separate object 172

Figure 6.1: A geometric analogy problem 178

Figure 6.2: Image pairs requiring perceptual reorganization 182

Figure 6.3: Strategy for finding differences between two images (see Appendix B for its implementation in the Spatial Routine language) 184

Figure 6.4: Geometric analogy problems requiring second-order comparison 187

Figure 6.5: The answer chosen depends on whether one prefers canonical reflections or rotations 189

Figure 6.6: Reflecting the "B" shape will produce an identical "B" shape, as in Figure 6.4B 190

Figure 6.7: The dot to be removed is in different locations in A and C 190

Figure 6.8: Problems 1-3 (times are seconds required for human participants to pick an answer; values below answers are the percentage of participants who picked each answer) 194

Figure 6.9: Problems 4-6 ... 195

Figure 6.10: Problems 7-9 ... 196

Figure 6.11: Problems 10-12 ... 197

Figure 6.12: Problems 13-15 ... 198

Figure 6.13: Problems 16-18 ... 199

Figure 6.14: Problems 19-20 ... 200

Figure 7.1: A matrix problem ... 209

Figure 7.2: Example problems for various Carpenter et al. (1990) rules ... 211

Figure 7.3: This matrix problem requires complex perceptual reorganization to solve ... 214

Figure 7.4: Two 1x1 matrix problems ... 217

Figure 7.5: Solution strategies for the above problems ... 217

Figure 7.6: Problems where computing the pattern of variance may be difficult ... 219

Figure 7.7: Subroutine for finding differences in a row (see Appendix C for its implementation in the Spatial Routine language). Continued in Figure 7.8 ... 221

Figure 7.8: Subroutine for finding differences in a row. Continued from Figure 7.7 ... 222

Figure 7.9: Different ways two textures might overlap ... 230

Figure 8.1: Oddity task problem from Dehaene et al. (2006). The image without parallel lines is the odd one out ... 242

Figure 8.2: C and D can only be solved by considering edges ... 244

Figure 8.3: Certain information can be filtered out ... 245

Figure 8.4: B and C require a different similarity measure ... 246

Figure 8.5: One of the six problems the model failed to solve. Average human performance: 68% (American adults), 86% (Mundurukú) ... 250

Figure 8.6: Problems with closed and open shapes ... 251

Figure 8.7: Problems relying on (A) shape transformation and (B) shape symmetry 251

Figure 8.8: Problems requiring shape comparison between images 256

Figure 8.9: Problems involving quantitative features 257

Figure 8.10: Problem in which the quantitative difference is particularly salient 258

Figure 9.1: Two image pairs. It is harder to align the same-shaped objects in B than in A..................... 260

Figure 9.2: Can you spot the difference between each image pair?... 261

Figure 9.3: Can you spot the difference between the more complex image pairs?.................................. 261

Figure 9.4: Object-level differences (A, B) may be more salient than edge-level differences (C)........... 262

Figure 9.5: Problems involving shape rotations.. 264

Figure 9.6: An oddity task problem (A) and a Raven’s matrix problem (B) with identical images 264

Figure 9.7: This geometric analogy problem requires quantitative information to solve 265

Figure 9.8: This geometric analogy problem presents a novel challenge 265

Figure A.1: Two images that might be compared... 289

LIST OF TABLES

Table 3.1: Orientation-invariant qualitative vocabulary for edges 84

Table 3.2: Additional orientation-specific vocabulary for edges 87

Table 3.3: Object-level qualitative vocabulary. Terms marked with an ‘O’ are orientation-specific 88

Table 3.4: Candidate inferences between images 10A and 10B 92

Table 3.5: Types of shape comparisons 102

Table 4.1: Pattern of variance for the first two images in Figure 4.6 129

Table 4.2: Generalized forms for predicates in difference representations 138

Table 4.3: A template and examples of operation calls 155

Table 4.4: A template and examples of control structures 160

Table 4.5: A spatial routine for solving Bongard problems 164

Table 4.6: Pattern of variance for Figure 4.20 165

Table 6.1: Linear model for human reaction times on geometric analogy 204

Table 7.1: Linear regression for accuracy of Northwestern students 233

Table 7.2: Second linear model for accuracy of Northwestern students 234

Table 7.3: Third linear model for accuracy of Northwestern students 235

Table 7.4: Linear model for reaction times of Northwestern students (in seconds) 235

Table 8.1: Accuracy of the model and each participant group on the 45 oddity task problems 248

Table 8.2: Correlations in accuracy on each of the 45 problems (Pearson’s r) 249

Table 8.3: Rankings of the 6 problems the model failed to solve (1 = easiest, 45 = hardest) 249

Table 8.4: Linear models for each group’s accuracy 253

Table 8.5: Linear models for each group’s accuracy, with Elems2 253

Table 8.6: Linear models for each group’s reaction times (in s) 254

1. Introduction

Spatial problem-solving tasks are a popular tool for evaluating people's cognitive abilities. For example, in geometric analogy (Figure 1.1), an individual is shown an array of images and asked "A is to B as C is to…?" Like all the tasks I am studying, geometric analogy requires: 1) building up representations of two-dimensional images; 2) comparing those representations; and 3) identifying a pattern across them. Thus, these tasks depend critically on one's ability to encode spatial relations within an image, compare images, and abstract out higher-order relations between then based on what is common or different.

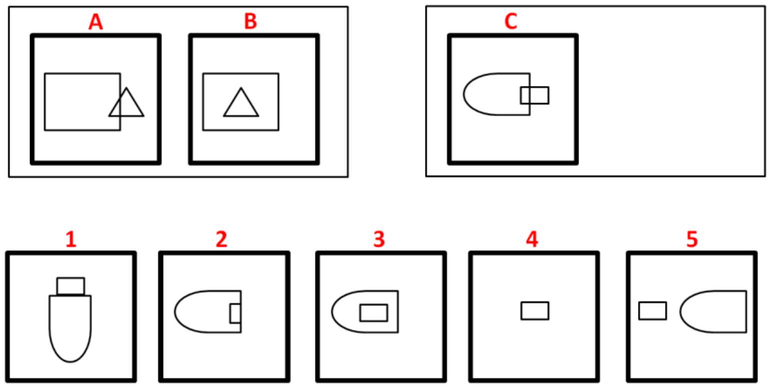


Figure 1.1 Geometric Analogy problem.

In the past, spatial problem-solving has been used to evaluate various abilities, from geometric knowledge (Dehaene et al., 2006) to general intelligence (Raven, Raven, & Court, 1998). However, there is disagreement about what exactly a particular task evaluates (e.g., geometric analogy: Sternberg, 1977; Mulholland, Pellegrino, & Glaser, 1980; Raven's Progressive Matrices: Carpenter, Just, & Shell, 1990; Primi, 2001). I believe we need a more concrete understanding of the representations and processes people use to solve these tasks. Applied to a single task, this might illuminate the abilities that separate one person's performance from another's. Applied across tasks, it should help explain how people reason about space and spatial relations.

For my thesis, I have constructed Spatial Routines for Sketches (SRS), a general framework for building, evaluating, and comparing spatial problem-solving task models. By *task model*, I mean an end-to-end model of human performance on a task, which begins with visual input, generates a representation, reasons over the representation, and chooses an output behavior. While the SRS models may vary in their specific strategies, they are all built upon three core hypotheses:

1) When possible, people use *qualitative*, or categorical, representations to reason about space (e.g., Biederman, 1987; Kosslyn et al., 1989; Forbus, Nielsen, & Faltings, 1991). In particular, they encode the qualitative spatial relations between elements in a visual scene.

2) Qualitative, structural representations of space are compared via structure-mapping (Gentner, 1983). According to structure-mapping theory, people compare two cases by aligning their common relational structure, thereby highlighting commonalities and differences across the cases.

3) Spatial problem-solving requires flexibly moving between different levels in a hierarchy of spatial representations (Palmer, 1977) until a suitable level is found. A level of representation is suitable if, when the representations are compared, a pattern emerges which can be used to solve the problem. For example, Figure 1.1 involves the spatial relations between the objects in each image. However, other problems might require focusing on the edges in a single object, or backing out and considering groups of objects.

I have two primary goals in modeling spatial problem-solving. The first is to evaluate whether the above hypotheses are sufficient for explaining human performance. A *sufficient* task model should meet two criteria: 1) The model should perform the task with a high degree of accuracy, making no more errors than a typical human would. 2) When the model does fail, its error patterns should match human error patterns, i.e., problems that are hard for the model should also be hard for people.

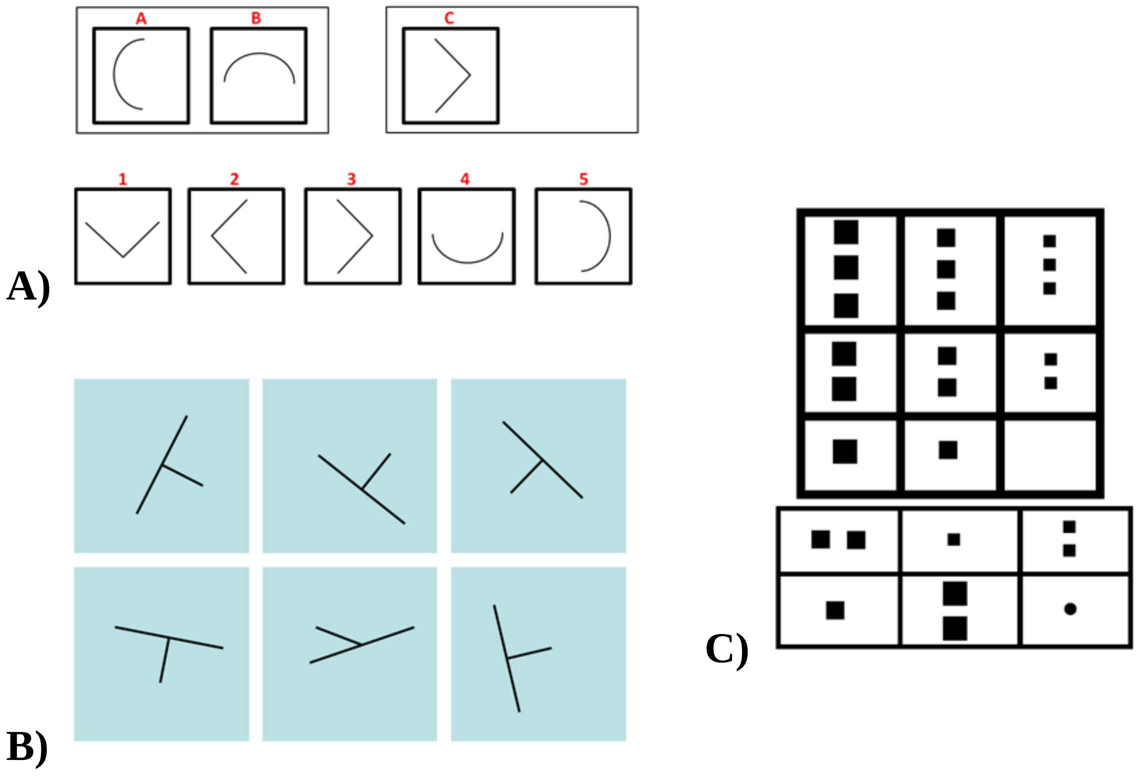


Figure 1.2 Three spatial problem-solving tasks.

A: Geometric Analogy: "A is to B as C is to…?"

B: Oddity Task: "Pick the image that doesn't belong.")

C: Raven's Progressive Matrices: "Pick the image that best completes the matrix." (not an actual test problem)

The second goal is to learn more about how people solve spatial problems. In particular: 1) What cognitive abilities make one person better at problem-solving than another? 2) What factors make one problem harder to solve than another? The models can help answer these questions only after the first goal is met: once there is evidence that a model's representations and processes align with humans, we can begin asking whether certain problems are harder because of the greater load they put on particular representations or processes.

I begin with my motivation for studying this class of tasks. I then outline the current understanding of the domain. After, I describe the key claims of my thesis, and then the particular contributions I have made.

1.1 Motivation

Figure 1.2 gives examples of three different tasks: geometric analogy, the oddity task, and Raven's Progressive Matrices. These three tasks are the focus of my thesis. Solving each of these requires reasoning intelligently about spatial relations between objects. In some cases, one must also reason about spatial *transformations* between objects. For example, Figure 1.2A requires recognizing that there is a spatial transformation, a rotation, between the two curves. Often, information about spatial relations and spatial transformations must be integrated when solving a problem.

One might question whether these abstract tasks relate to human functioning in the real world. However, I believe they tap into the same cognitive abilities that people to use to reason about space, and relations in general, in their everyday lives. There are two bodies of work supporting this claim: 1) studies on spatial visualization ability, and 2) studies on general problem-solving ability.

Spatial Visualization

Spatial visualization (McGee, 1979) is the ability to manipulate mental representations of images. Spatial problem-solving tasks often require this for computing transformations between shapes. For example, in Figure 1.2A one must mentally rotate one shape to align it with the other (Shepard & Metzler, 1971). Spatial visualization is evaluated more directly in other tasks, such as the paper-folding task (Snow, Kyllonen, & Marshalek, 1984), where an individual is shown a paper with dots on it and asked to determine which of several other images portrays a valid result of folding the paper.

Spatial visualization is an important skill in many professions, including geoscience, meteorology (Trickett et al., 2007), engineering, surgery (Hegarty et al., 2007), and dentistry (Just, 1979). All of these occupations require that individuals be skilled at mentally manipulating representations of images. For example, when a surgeon is operating, they may be required to mentally construct a three-dimensional structure from a two-dimensional image.

Importantly, real-life spatial visualization ability can be evaluated by abstract problems like paper-folding. Several studies have found correlations between performance on abstract tasks and real-world ability in surgeons (see Hegarty et al., 2007, for a review). Shea, Lubinski, & Benbow (2001) found that performance of 13-year-olds on spatial visualization tasks predicted what they would study in college and what eventual job they would get, even controlling for verbal and mathematical ability. Individuals with higher spatial ability were more likely to be scientists and engineers. This analysis was restricted to the top 1% of 13-year-olds, but ongoing research (Hedges & Chung, in preparation) suggests the findings generalize to the rest of the population.

Thus, it appears that abstract, spatial problem-solving tasks tap into a spatial visualization ability that is useful in many advanced disciplines. If we can understand how people perform spatial transformations in these tasks, we will be better prepared to educate students in spatial skills, and hopefully enhance their ability to perform in those disciplines.

General Problem-Solving Ability

Many problem-solving tasks in the spatial, verbal, and mathematical domains have been used to evaluate people's abilities. In the past, correlations in performance across these tasks caused Spearman (1923; 1927) to put forward the notion of *g*, a single, general intelligence measure which could predict an individual's ability across a large set of task domains. While this thesis is not directly concerned with *g*, Spearman's work suggests that some mental abilities are utilized across a wide range of tasks. If spatial problem-solving tasks tap in those abilities, then our models can give us insights into people's general reasoning processes.

In fact, there is strong evidence that one of the tasks, Raven's Progressive Matrices (RPM), *does* tap into general abilities. RPM was originally designed to evaluate one major component of *g*, *eduction*, or the ability to identify meaningful patterns in confusing data (Raven, Raven, & Court, 1998). Since the test's creation, several studies have shown it to be one of the highest single-test correlates with *g* (e.g., Burke & Bingham, 1969; Zagar, Arbit, & Friedland, 1980; see Raven, Raven, & Court, 2000b, for a review), meaning an individual's performance on RPM predicts their performance on many other intelligence tests. In a multi-dimensional scaling analysis of ability tests, Snow and colleagues (Snow, Kyllonen, & Marshalek, 1984; Snow & Lohman, 1989) found that RPM lies in the middle of the ability space. That is, while most ability tests cluster with other tests in the same domain, e.g., verbal, mathematic, and spatial tests, RPM correlates highly with the most abstract ability tests from each domain. This suggests that although RPM is a visuo-spatial task, it evaluates several domain-general mental abilities. My hope, in modeling this task, is to gain a greater understanding of what those abilities are.

1.2 Background

Two of the spatial problem-solving tasks, geometric analogy and the Raven's Progressive Matrices, have been studied extensively in the past, while research on the oddity task is more limited. In the following sections I summarize the research, focusing on two questions: 1) What are the processes people use to perform the task? 2) What factors contribute to the relative difficulty of each problem?

Geometric Analogy

Processes

Geometric analogy (Figure 1.1, Figure 1.2A) has been studied and modeled by many researchers over the last 50 years. Evans' ANALOGY (1968) was the first computational model of the task. A sophisticated program for its time, ANALOGY automatically encoded representations of the images in a problem and could even identify spatial transformations between shapes, such as rotations. Once representations had been constructed or hand-coded, it solved a problem via what has been termed an *infer-infer-compare* strategy (Mulholland, Pellegrino, & Glaser, 1980):

1) Infer a mapping between images A and B, describing what changed between them.

2) For each possible answer *n*, infer a mapping between images C and *n*.

3) Compare the A/B mapping to the C/*n* mapping for each possible answer, and choose the answer for which this comparison returns the closest match.

ANALOGY's mapping processes are unlike human mapping processes in that they perform an exhaustive search for the best possible mapping, rather than using heuristics (Gentner, 1983). However, the infer-infer-compare model remains a reasonable strategy for geometric analogy, and it is one we will return to.

Sternberg (1977) argued that people actually solve problems like geometric analogy via an *infer-map-apply* strategy (Mulholland, Pellegrino, & Glaser, 1980):

1) Infer a mapping between images A and B, describing what changed between them.

2) Compute a mapping between images A and C, identifying their corresponding elements.

3) Based on this mapping, apply the A/B differences to C to compute D', a representation of the image that would best complete the analogy. This D' representation can then be compared to each possible answer to see which one best matches it.

Sternberg produced evidence that people utilized infer-map-apply on his problem set, while Mulholland, Pellegrino, & Glaser (1980) argued that people utilized infer-infer-compare on *their* problem set. Later, researchers suggested that people adjust their strategy depending on factors such as the problem's complexity and the relatedness of the items being compared (Bethell-Fox, Lohman, & Snow, 1984; see also Grudin, 1980, on verbal analogies).

These early researchers were focused on the particular strategy, or the set of cognitive components, that a person would use to solve a geometric analogy. However, they were less clear on what specific mechanisms might be used, for example, to compare images A and B. Only Evans (1968) had a complete computational model. While computational models of geometric analogy have become more popular in recent years (e.g., Bohan & O'Donoghue, 2000; Schwering et al., 2009), I believe no other researchers have built an end-to-end computational model that could be compared against human performance.

Factors Contributing to Difficulty

The above researchers also analyzed the factors contributing to each problem's relative difficulty. In generating their stimuli, Mulholland, Pellegrino, & Glaser (1980) independently varied the number of elements in each image and the number of transformations between images, e.g., the number of differences between images A and B. They found that as either number of elements or number of transformations increased, the reaction time increased, while the accuracy decreased. Importantly, for reaction time the effects of elements and transformations were not simply additive; there was a particularly great cost when both number of elements *and* number of transformations increased. The researchers believed this was due to working memory load. As the number of elements or transformations increased, the working memory load also increased. For particularly large numbers of both, working memory load exceeded people's capacity, forcing them to change their problem-solving strategy, and resulting in substantial reaction time costs.

Bethell-Fox, Lohman, & Snow (1984) independently varied several factors in their geometric analogy problems: number of elements in the images, number of transformations between images, figural vs. spatial transformations, number of possible answers to choose from, and similarity of distractors to the correct answer. They found that all these factors correlated with reaction times, while all factors except number of transformations correlated with error rates. They also found some interesting interactions between the factors—for example, those problems with the most images *and* the most possible answers were significantly harder than other problems. Again, this may have been because the large memory load required a strategy shift.

Bethel-Fox et al. found that problems involving spatial transformations, e.g., rotations between shapes, were more difficult and required more time to solve. These are, of course, the problems which evaluate spatial visualization ability. Thus, the results suggest that spatial visualization, and the computation of spatial transformations, requires the deployment of additional resources.

Overall, these results suggest at least two factors contributing to difficulty on geometric analogy problems: working memory load and spatial transformation. There is limited support for a third factor: an individual's ability to plan out, evaluate, and modify a problem-solving strategy. This factor might be important for problems where the memory load is particularly high, requiring a strategy shift.

Raven's Progressive Matrices

Processes

Carpenter, Just, and Shell (1990) conducted a thorough analysis of how people solve Raven's Progressive Matrices (RPM) (Figure 1.2C). After detailed studies of human performance on the Advanced Progressive Matrices (APM), the hardest version of RPM, they built two computational models capable of taking the test. These models, though powerful, operated only on hand-coded input; they were unable to process the visual stimuli shown to human test-takers.

Both models were based on the observation that people solve these problems by examining the first row of the matrix, identifying corresponding elements in each image of the row, and determining a rule to describe the pattern of variation across the corresponding elements. The experimenters identified five abstract rules that were sufficient for solving all but two problems in APM. These rules, in order of difficulty, are:

1) Constant in a row: The elements are the same.

2) Quantitative pairwise progression: Going across the row, the element gradually changes in some way, e.g., becoming bigger, or rotating.

3) Distribution of three values: There are three different elements in the three images of the row. Every row must contain each of those three elements, but their order varies.

4) Figure addition or subtraction: If the first image contains element X and the second image contains element Y, the third image must contain elements X *and* Y. Subtraction is similar.

5) Distribution of two values: There is an element that is present in two of the images but not the third.

For example, to solve Problem 2B, one would study the images in the first row, note that the groups of three squares correspond to each other, and recognize that the relation between these groups can best be described by a *quantitative pairwise progression* rule: the squares are becoming smaller. One would then select the answer that best fit this rule in the bottom row.

Carpenter, Just, and Shell's (1990) first model, FAIRRAVEN, could perform APM about as well as the average participant from their subject pool. The second model, BETTERAVEN, performed at the level of the best participants. BETTERAVEN performed better due to a couple key changes: 1) BETTERAVEN could identify cases where there was an element in only two of the three images in a row, meaning it could use the *distribution of two values* rule; 2) BETTERAVEN had better goal management. It identified candidate rules one at a time, and it backtracked when a candidate rule proved ineffective for solving a problem.

Based on the difference between the models, the experimenters suggested that the ability to manage goals is a key factor in solving the hardest problems. They saw this as being linked to working memory: a complex problem requires that one manage a hierarchy of goals, and keeping them all in memory can be quite difficult.

In their analysis, the researchers noticed another interesting difference between high- and low-performers on the task. In some cases, there was more than one rule that might be applied to solve a problem. For example, suppose the three corresponding elements were an arrow pointed up, an arrow pointed right, and an arrow pointed down. This set of elements could be seen as either a quantitative pairwise progression, in which an arrow shape gradually rotates, or a distribution of three, with three different shapes. While either of these rules is sufficient, the first rule is better, as it is more compact. That is, one doesn't have to remember all three shapes individually. On problems such as these, verbal protocols showed that higher performers used the quantitative pairwise progression rule more often than lower performers.

Carpenter et al. suggested the above result occurred simply because higher performers were more consistent about looking for the simpler rules before they looked for the more complex rules. However, I think it is likely that all participants looked for the simpler rules first. I interpret these results as suggesting, rather, that higher performers are better at identifying more abstract relationships between elements. Whereas the lower performers saw the arrows as different shapes, the higher performers were better able to compare the arrows, perform a spatial transformation, and identify a rotation between the arrows' shapes. Identifying more abstract relations on problems like these would reduce working memory load and generally aid in solving the problems.

Carpenter et al. failed to address a couple important components of human processing with their models. Firstly, the models did not generate representations from the images; all representations were hand-coded based on participant descriptions. The experimenters suggested the visual encoding process was irrelevant, given how well performance on the task correlates with other, non-visual tasks. However, this assumes visual encoding ability does not generalize to encoding ability in other modalities. I believe RPM taps into a general ability to encode a stimulus at the appropriate level of abstraction. Indeed, the authors themselves suggest that an ability to decrease working memory load by using more abstraction representations might explain the correlation between RPM and the Towers of Hanoi problem, a very different task

Secondly, the FAIRAVEN and BETTERAVEN models did not learn the five rules described above. Rather, abstract forms of all the rules were hard-coded into the models. Thus, the models fail to explain how individuals begin with basic comparisons of pairs of images and develop, over time, an understanding of a complex rule that holds across a row of images.

Factors Contributing to Difficulty

Two groups of researchers (Vodegel-Matzen, van der Molen, & Dudink, 1994; Embretson, 1998) have evaluated Carpenter, Just, and Shell's (1990) analysis by creating their own, experimental matrix items in which they varied the number and type of Carpenter rules. Both groups found that a rule complexity measure, based on the number and difficulty of rules, correlated highly with problem difficulty: problems with more rules, and with more difficult rules, are harder to solve. This is likely because such problems both put a greater load on working memory and require more sophisticated problem-solving techniques. I believe a computational model which makes those problem-solving techniques explicit is needed to gain a better grasp on what exactly makes more complex problems harder.

Primi (2001) designed new matrix items in which he independently varied the number of elements in each image, the number of rules describing a row of images, the complexity of rules, and *perceptual organization*. Perceptual organization referred to how difficult it was to identify the corresponding elements in a row of images. In his *low perceptual organization* problems, the correspondence-finding component was made difficult through misleading cues that encouraged test-takers to put the wrong elements into correspondence. Perhaps unsurprisingly, he found that difficulty of correspondence-finding was the greatest predictor of problem difficulty.

The official RPM problems, unlike the problems in Primi's test set, were not explicitly designed to make correspondence-finding difficult. Nonetheless, there are some problems which invite inappropriate correspondences (Carpenter, Just, & Shell, 1990). In such cases, I suspect the most important skill is an ability to backtrack and look for alternate mappings when a set of correspondences prove insufficient for solving a problem. Thus, dealing with incorrect correspondences, like dealing with excessive memory load, may require an ability to evaluate and dynamically modify problem-solving strategies.

Oddity Task

Processes

The oddity task (Figure 1.2B) is a set of 45 problems which Dehaene et al. (2006) constructed to evaluate people's geometric knowledge. The experimenters hypothesized that the problems tap into *core knowledge of geometry* (Spelke & Kinzler, 2007), an innate, universal cognitive module. This module would presumably understand key geometric concepts like perpendicular lines, and would therefore be used to solve problems like Figure 1.2B.

To test the universality of geometric knowledge, Dehaene et al. gave their 45 problems to participants of varying ages from two cultural groups: North Americans and the Mundurukú, a South American indigenous group. They found that the Mundurukú performed above chance on nearly all the problems, despite their lack of formal schooling in geometry and mathematics. Furthermore, there was a significant correlation between the American and Mundurukú error patterns. The experimenters took this as evidence that the two groups were working with a shared, universal set of core geometric knowledge.

As this work is relatively new, no other researchers have explored the processes people might utilize in performing the task. However, I have suggested people likely use processes similar to those used in the previous tasks (Lovett, & Forbus, 2011). Whereas geometric analogy involves comparing two images and noticing their differences, the oddity task involves comparing images and noticing their commonalities. Later, I will argue that both of these comparisons can best be modeled as structure-mapping processes (Gentner, 1983).

Factors Contributing to Difficulty

The oddity task problems (Dehaene et al., 2006) were designed to evaluate understanding of two-dimensional spatial concepts. As such, they vary greatly in the type and complexity of the spatial concepts they test for. Several of the more difficult problems involve rotations or reflections between shapes. Thus, as with geometric analogy, people may have more difficulty with problems that require spatial visualization.

Importantly, these problems do not merely rely on one's ability to mentally rotate shapes. They also require that one notice that there is a possible shape transformation, and then put in the effort to compute the transformation. Thus, there may be a motivational element. Recall that on the RPM, higher-performing participants were more likely to notice transformations like a rotation of an arrow shape, even when the transformation wasn't strictly necessary for solving the problem. I believe that across all three tasks, recognition of spatial transformations depends on: 1) spatial visualization ability, and 2) a general intellectual interest in comparing elements and looking for more abstraction relationships between them.

1.3 Claims

My thesis rests on two broad sets of claims. The first set is about the representations and processes we can use to model human spatial problem-solving. The second set is about what we can learn from these models, in terms of the factors that contribute to a problem's difficulty and the processes that explain variation in human performance.

Modeling Spatial Problem-Solving

The studies described above tell us a little about the processes, and less about the representations, that people utilize during spatial problem-solving. There are several open questions that must be resolved to model the tasks from end to end. I propose the following hypotheses about human representations and processes:

1) When possible, people utilize *qualitative* representations of the spatial relations between elements in an image (e.g., Biederman, 1987; Kosslyn et al., 1989; Forbus, Nielsen, & Faltings, 1991) when reasoning about space.

2) These qualitative representations are hierarchical (Palmer, 1977). That is, one can reason about relations between objects in an image, or one can focus in and reason about relations between parts of an individual object, or one can zoom out and reason about relations between groups of objects.

3) Qualitative, relational representations are compared via structure-mapping (Gentner, 1983, 1989), a domain-general process of structural alignment. Structure-mapping can identify commonalities and differences in two representations, or it can determine their similarity.

4) The computation of spatial transformations, such as rotations between shapes, is accomplished via an interaction between hierarchical representations and structure-mapping. To compute a transformation between two shapes, an individual uses structure-mapping over representations of the shapes' parts to identify corresponding parts. Then, a spatial transformation is applied to put the corresponding parts into alignment.

5) Spatial problem-solving requires control processes which can look over a problem, apply a strategy—where a strategy consists primarily of a set of comparisons via structure-mapping—and, when a strategy fails, backtrack and attempt a new strategy.

In the following subsections, I briefly justify each hypothesis and then outline how it can be modeled.

1) **Qualitative Representations of Spatial Relations**

There is good evidence that people are sensitive to the qualitative, or categorical, relations between objects in a visual scene. For example, parallel lines (Abravanel, 1977) and concave corners between edges of a shape (Bhatt et al., 2006) are particularly salient to people, suggesting that we make a qualitative distinction between parallel and non-parallel, or between concave and convex. In some cases, people go out of their way to identify a qualitative relation between objects: when people are asked to memorize the location of a dot in a circle, there is evidence they mentally divide the circle into four quadrants and then qualitatively encode which of the four quadrants the dot was located in (Huttenlocher et al., 1991).

Qualitative spatial relations are particularly useful in spatial problem-solving because they abstract out irrelevant information, while often maintaining the important features of a visual image. For example, the geometric analogy problem in Figure 1.1 is most easily solved if one distinguishes between two qualitative topological relations: *overlapping* and *containing*. Similarity, the oddity task problem in Figure 1.2B is most easily solved if one encodes a qualitative relation for perpendicular lines, while ignoring the lines' absolute orientations.

I propose that, when possible, people utilize qualitative representations during spatial problem-solving. That is, once they have built a qualitative representation of an image, they will ignore quantitative data, such as the absolute locations or orientations of the elements in the image. Quantitative information will be used only when it is necessarily (as when computing spatial transformations; see below).

I model the computation of qualitative spatial relations with CogSketch (Forbus et al., 2011), a sketch understanding system. Given a two-dimensional sketch, CogSketch automatically computes qualitative relations between the objects in the sketch.

2) **Hierarchical Representations**

In vision science, the study of perceptual organization is concerned with how people divide a visual scene into a coherent set of objects. It is generally assumed (e.g., Ullman 1987; Palmer and Rock 1994) that bottom-up processes create a set of *entry-level* units. These initial elements can then be grouped into larger objects or parsed into smaller objects. A conclusion one may draw from this is that the distinction between the objects in a scene and the parts in an object is a difference of degree only; one of these may be represented and reasoned over as easily as the other. Indeed, Palmer (1977) suggested that people possess hierarchical representations of visual space. At any level of abstraction, there is a set of *structural units* (SUs), and there are relations between the SUs. Zooming in on any single SU, one can identify the set of SUs and relations that make it up.

I propose that such hierarchical representations play a crucial role in spatial problem-solving. To solve any spatial problem, one must determine the appropriate level of abstraction: should one be reasoning over objects, object parts, or object groups? For example, the oddity task problem in Figure 1.3 is easiest if one thinks about groups of dots, rather than individual dots.

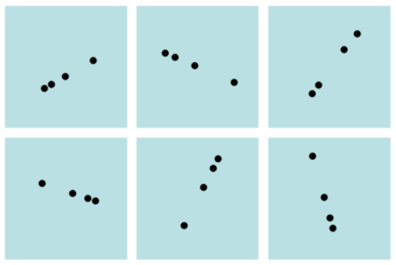


Figure 1.3 An oddity task problem involving groups of objects.

In my models I make the simplifying assumption that there are three hierarchical levels for two-dimensional images: objects, groups of objects, and edges of individual objects. CogSketch only computes relations between objects. Therefore, I have developed the Perceptual Sketchpad, an extension to CogSketch which can automatically segment an object into its edges, or combine several objects to form groups. At either of these levels, it computes qualitative spatial relations between the elements.

3) **Comparison via Structure-Mapping**

Structure-mapping is a theory of relational comparison first proposed to explain how people perform analogies (Gentner, 1983, 1989). According to structure-mapping, people compare two relational representations by aligning their common relational structure. This process highlights both their commonalities and their differences, particularly *alignable differences*, i.e., differences between elements in corresponding locations in the aligned structure. According to structure-mapping theory, people strive to find the most *systematic* mapping, the mapping that aligns the most and the deepest common structure.

Structure-mapping has been modeled computationally using the Structure-Mapping Engine (SME) (Falkenhainer, Forbus, & Gentner, 1989). Given two relational descriptions, a *base* case and a *target* case, SME computes one or more *global mappings* between them. Each mapping consists of: 1) *correspondences* between elements in the two cases; 2) a *structural evaluation score*, a measure of the similarity of the cases based on the systematicity of the mapping; and 3) *candidate inferences*, inferences about the target based on elements in the base that failed to map to the target. Candidate inferences can be used to identify differences between the base and the target.

While structure-mapping was originally proposed to explain abstract analogies, there has been mounting evidence that it can also explain people's concrete comparisons of visual stimuli (Markman & Gentner, 1996; Lovett et al., 2009a; Sagi, Gentner, & Lovett, in press). I believe it may play a ubiquitous role in spatial problem-solving because of its usefulness at various stages in the problem-solving process. For example, the infer-infer-compare strategy for geometric analogy, described above, relies on both computing the differences between images and comparing two sets of differences. SME can do both these things, as the sets of differences computed by SME can themselves be compared via SME in a second-order comparison (Lovett et al., 2009b).

4) **Structure-Mapping in Spatial Transformations**

One of my goals is to better understand how people perform spatial visualization, particularly how they compute transformations between shapes. Work on *mental rotation* (Shepard & Metzler, 1971; Shepard & Cooper, 1982), has shown that the time required to determine that one shape is a rotation of another is proportional to the degrees of rotation between them. This suggests that people perform this task by mentally rotating their representation of one shape to align it with the other. If so, people cannot be working solely with a qualitative shape representation, as described above. Qualitative representations would not include the absolute orientations of shape parts, so there would be no need to transform them in an analog fashion.

I believe mental rotation is performed on a quantitative, orientation-specific shape representation. While the exact form of this representation is unclear, it can be modeled simply as the set of edges in a shape, along with each edge's quantitative orientation, length, and location. However, a mystery remains. Typically, the time to perform a mental rotation is proportional to the degrees of rotation along the *shortest* possible rotation between the two shapes. How do people know which way to rotate one shape to quickly align it with the other?

To answer this question, I have proposed a three-stage model of mental rotation:

1) Compare qualitative edge-level representations via structure-mapping. Identify the corresponding edges.

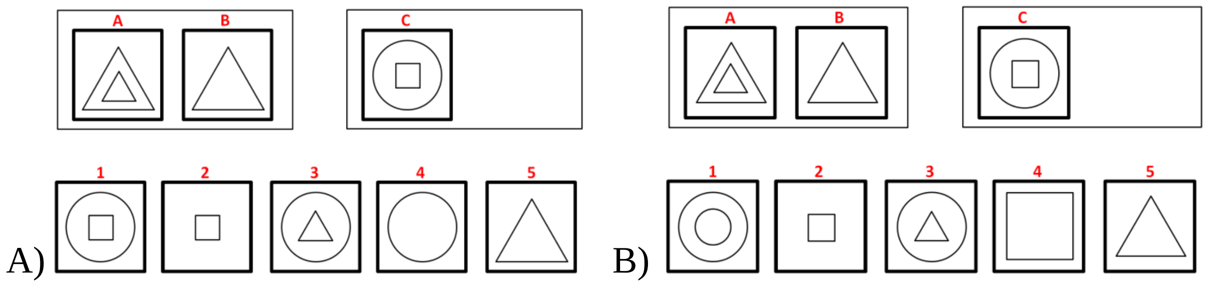
2) Take a single pair of corresponding edges and calculate the shortest possible rotation between them.

3) Apply this rotation to the full set of quantitative edges in the first shape. Evaluate whether this aligns those edges with the edges in the second shape.

This approach can identify other transformations, such as reflections, as well. I describe the approach in detail in Chapter 3.

5) **Control Processes**

I have argued that structure-mapping can account for much of the processing in spatial problem-solving. However, there must also be a control process which chooses a strategy, oversees the application of that strategy, and selects a new strategy if the original fails to produce a solution. New strategies may be necessary when a particularly complex problem overloads the working memory (Mulholland, Pellegrino, & Glaser, 1980; Bethell-Fox, Lohman, & Snow, 1984), or when a tricky problem causes one to identify the wrong corresponding elements (Primi, 2001).



Average RT: 6.7 s

Accuracy: 100%

Average RT: 26.7 s

Accuracy: 56%

Figure 1.4 Two related geometric analogy problems, along with average reaction times and overall accuracies for human participants.

For example, consider the geometric analogy problems in Figure 1.4. While these problems are nearly identical, the second one is significantly harder (Lovett et al., 2009b) because the obvious mapping between images A and B, in which the large triangle in A corresponds to the large triangle in B, is incorrect. Thus, the problem requires backtracking and identifying an alternate, less obvious mapping, in which the small triangle in A corresponds to the big triangle in B. Clearly, this type of backtracking is difficult for people.

I have built control processes directly into Spatial Routines for Sketches. A spatial routine can include loops that iterate through multiple strategies until a sufficient solution is found—a sufficient solution may be one whose SME mapping receives a reasonably high score, or one whose SME comparison contains no differences. Spatial routines can also contain nested loops, allowing multiple combinations of strategies to be attempted. Thus, a spatial routine is much like a computer program (Ullman, 1987).

Note that I do not model working memory capacity. Because there are no limits to a model's working memory, it will never fail to solve a problem because too many things must be kept in memory. Similarly, it will never forget the previously completed steps of a problem and need to rework them. I view the modeling of working memory capacity as future work which can be attempted when spatial working memory is better understood.

Using the Models to Study Human Performance

Once models of spatial problem-solving have been developed, we can use them to study human performance on the tasks, identifying the factors that make one problem easier or harder than another. While several other researchers have considered this issue, an end-to-end task model allows us to be much more explicit about how a particular factor puts more load on the model's representations and processes. The models can also be used to automatically tag each problem with information such as the number of elements that must be represented, the number of operations that must be applied to solve the problem, etc. Thus, a model can code a problem more objectively than if we simply hand-code each problem stimulus based on our best guesses about what the problem requires.

Once problems are coded for different factors, we can build mathematical models of performance matching particular individuals, or groups, in order to determine how individuals vary in their ability to handle different sources of difficulty. In this way, we can hopefully develop a better understanding of commonalities and differences in people's spatial reasoning ability.

In my analysis, I focus on three broad classes of factors: encoding & abstraction, working memory load, and control processes. I describe each of these next.

Encoding & Abstraction

Encoding & abstraction refers to the ability to encode representations at the appropriate level of abstraction for solving a problem. Some problems may be more difficult because they require representations that people are less inclined or less able to compute. I am interested in two forms of abstraction, which I call *entity abstraction* and *relational abstraction*.

Problems that are high on entity abstraction require the solver to encode elements at a particular level in the representational hierarchy. This might be either a larger scale or a smaller scale than the most comfortable level for an individual. To demonstrate, let us consider a simple form of analogical problem-solving: the letter string analogy (Mitchell, 1993; Hofstadter, 1995). The simplest letter string analogy is:

abc : abd :: rst : ?

Here, the obvious solution is "rsu," based on incrementing the final letter by one. Suppose instead we have:

abc : abd :: rrsstt : ?

While an obvious solution here would be "rrsstu," a better one might be "rrssuu." Discovering this solution depends on entity abstraction. One must recognize that the appropriate level of abstraction for representing the "rrsstt" letter string is as two-letter groups. The entire "tt" group in "rrsstt" maps to the letter "c" in "abc."

On the other hand, problems that are high on *relational* abstraction require the solver to notice more complex relations between elements. Often, this involves comparing elements within a single image (or letter string) to recognize commonalities and differences among them, before one compares across images (or letter strings). For example, consider the problem:

abc : abd :: hkkuuu : ?

Here, an obvious solution might be to apply the letter successor relation and get "hkkuuv" or "hkkvvv." However, if one spends more time considering the relationships between the elements in "hkkuuu," one should recognize that this letter string contains a new kind of successor relationship: group length. One might then choose to increment this successor relationship instead, to get "hkkuuuu."

Of course, entity and relational abstraction occur in slightly different forms in spatial problem-solving. Entity abstraction is the ability to build a representation at the level of groups of objects, objects, or parts of an object, as the problem demands. Relational abstraction is the ability to recognize complex relations, particularly spatial transformations such as rotations or reflections. I use the letter string examples to illustrate that these abilities are not specific to the domain of spatial problem-solving. Entity and relational abstraction may rely on a general inclination to try out different abstractions for representing stimuli.

Unfortunately, my analysis is insufficient for distinguishing between the *inclination* and the *ability* to form abstractions. Thus, relational abstraction refers to both an interest in comparison and abstraction, and an ability to compute spatial transformations between shapes, i.e., spatial visualization ability.

The simplest way to determine whether a certain problem requires entity or relational abstraction is via ablation. That is, one can temporarily remove the model's ability to generate representations at a particular level, or to compute spatial transformations, and check whether the model now fails on the problem.

Working Memory Load

Working memory load is simply the number of things that must be kept in mind when solving a problem. Several studies have suggested that people are slower and less accurate to solve problems that involve more elements or more transformations between elements (e.g., Mulholland, Pellegrino, & Glaser, 1980; Bethell-Fox, Lohman, & Snow, 1984; Embretson, 1998; Vodegel-Matzen, van der Molen, & Dudink, 1994). One advantage of computational models is that they can automatically code the number of elements, number of relations, number of differences between representations, etc, that are used to solve a problem. Thus, we can easily code a problem for working memory load. We can evaluate to what extent these different factors (number of elements, number of relations, etc) place a load on working memory by considering which factors correlate with problem difficulty.

Control Processes

The articles cited above argue that more complex problems are harder only because of increased working memory load. However, an alternate explanation is that, in some cases, more complex problems require more careful control over problem-solving strategies. For example, in Raven's Progressive Matrices, as the number of elements increases, the problem of identifying the corresponding elements becomes more difficult. Thus, one is more likely to identify incorrect corresponding elements on the first try, fail to solve the problem, and require backtracking.

Carpenter, Just, and Shell (1990) have referred to this problem as goal management and have suggested that goal management itself boils down to working memory, since more complex problems require keeping a larger hierarchy of goals in memory. However, I believe we should consider control processes as a separate factor, particularly as the computational models allow us to directly code the number of operations and the amount of backtracking required to solve a problem. Thus, we can look at whether this factor explains any of the variance in problem difficulty, beyond what working memory load explains.

1.4 Contributions

My thesis consists of five contributions. Firstly, the Perceptual Sketchpad is an extension to the CogSketch sketch understanding system. Given a set of objects drawn in CogSketch, the Perceptual Sketchpad automatically segments each object into its edges and groups objects together based on similarity. It generates human-like qualitative representations at three hierarchical levels: groups, objects, and edges.

Secondly, Spatial Routines for Sketches (SRS) is a modeling framework inspired by Ullman's (1987) *visual routines*. As with visual routines, SRS utilizes a set of cognitive operations that we know people can accomplish. These include perceptual encoding and comparison via SME. The operations can be combined to create a spatial routine for performing some spatial task. For example, a spatial routine could describe one strategy for solving geometric analogy problems. A spatial routine is analogous to a computer program: each operation takes an input, processes it, and produces an output.

SRS allows researchers to construct cognitive models of human performance on a task. Once a researcher has written a routine, SRS implements its operations, allowing the routine to be tested on visual input. Psychological stimuli can be imported into CogSketch and processed by the spatial routine. Thus, a researcher can compare their model to human performance on the same stimuli.

The other contributions are the three task models constructed in SRS: geometric analogy, Raven's Progressive Matrices, and the oddity task.

1.5 Evaluation

I evaluate my claims using the three task models.

Geometric analogy is of interest because it has been modeled extensively by other researchers over the years, making it a good starting point. I evaluate the model by comparing it to human performance on a set of 20 problems originally used by Evans (1968) to evaluate his early analogical model.

Raven's Progressive Matrices is of interest because of the large body of work suggesting performance on the task correlates with performance on other measures of human intelligence (Raven, Raven, & Court, 2000b; Snow & Lohman, 1989). I evaluate the model on the Standard Progressive Matrices, a set of 60 problems.

The oddity task is of interest because Dehaene et al. (2006) used it to compare the geometric knowledge of two wildly different cultural groups: North Americans, and the Mundurukú. Thus, I can evaluate the model's ability to identify differences between cultural groups. This task consists of 45 problems.

For all three tasks, my first goal was to demonstrate that the models' representations and processes are sufficient for modeling human performance. Firstly, a model's overall accuracy on a task should be at least as high as the typical adult. Secondly, the model's error patterns should match human error patterns: problems that are difficult for the model should also be difficult for people.

My second goal was to identify sources of difficulty on each task. The models can automatically code each problem for items from the three classes of difficulty factors: encoding & abstraction, working memory load, and control processes. I use linear regression to determine how well these factors account for human accuracy and reaction times. In this way, the models can help us better understand what makes a problem easy or difficult for people.

Finally, for the oddity task only, I have built separate linear models for the different cultural and age groups. These models highlight some commonalities and differences in the representations and processes people use to reason about space.

1.6 Outline of the Thesis

In the following two chapters, I consider the key problems of visual encoding and comparison. I begin by outlining the psychological evidence for *hierarchical hybrid representations*, containing both qualitative and quantitative features at multiple levels of abstraction. I show how even basic visual comparisons require a strategic search through this hierarchy. I then describe my implementations of perceptual encoding, image comparison, and shape comparison.

In Chapter 4, I present Spatial Routines for Sketches (SRS), a general framework for modeling spatial problem-solving. I enumerate its key operations, including perceptual encoding, visual comparison, and visual inference.

In Chapter 5, I give an introduction to cognitive modeling with SRS, discussing the general principles that go into all my task models.

In Chapters 6-8, I describe the task models for geometric analogy, Raven's Progressive Matrices, and the oddity task. I present results comparing each model against human performance.

In Chapter 9, I discuss predictions that follow from my models. Several hypotheses and assumptions went into the production of each model, and these can be explicitly tested in future psychological studies.

Finally, in Chapter 10, I draw conclusions, discuss related work, and consider directions for future research.

2. Background

Spatial problem-solving depends on the ability to compare images and identify similarities and differences. This is difficult because even simple images contain many features, of which only a few are critical to a given comparison. For example, Figure 2.1A displays an oddity task problem, in which one must choose the image that doesn't belong. Each image contains four circles. The circles possess various features (e.g., roundness), and there are several spatial relations between the circles. On the other hand, each image also contains a row of dots. In five of the images, there is only a single row, whereas the last image contains a row plus an additional outlying dot. Thus, it is much easier to pick the odd image out if we view them as rows then if we view them as individual circles. However, we have no way of knowing a priori what features will be important when we consider a problem.

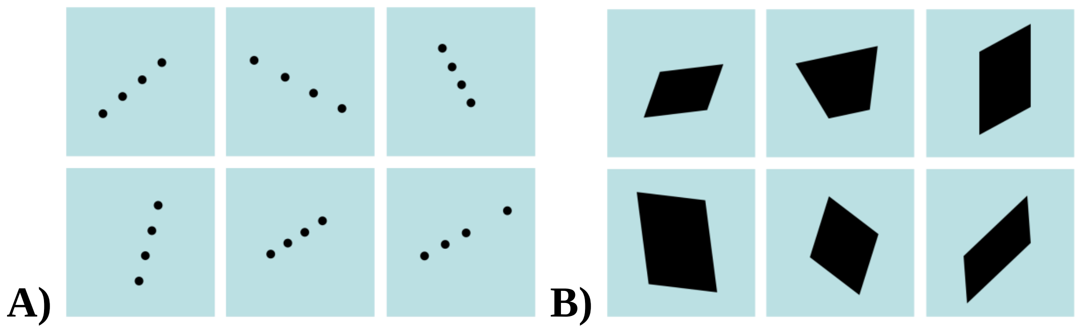


Figure 2.1 Two oddity task problem from Dehaene et al. (2006). Pick the image that doesn't belong.

To solve these problems, one must flexibly move between candidate representations until a comparison's critical features are discovered. Here, I argue that human perceptual systems produce *hierarchical hybrid representations* (HHRs) of space. These representations are hierarchical in that a given stimulus can be encoded at several levels of abstraction. They are hybrid in that, at each level, there are at least two types of representations: qualitative and quantitative. For example, consider the edges of the objects in Figure 2.1B. These edges possess several quantitative features, such as length and orientation, that are irrelevant to the problem. However, if we focus on the *qualitative* relations between edges, we see that in all images except one, there are two pairs of parallel edges.

Psychological evidence suggests hierarchical representations are explored top-down (Hochstein & Ahissar, 2002). That is, an individual will first compare two stimuli at a high level of abstraction and then move down as needed. In contrast, the temporal interaction between qualitative and quantitative representations is less clear (Bornstein & Korda, 1984; Kosslyn et al., 1977). However, in both cases, individuals may strategically reprioritize representations, depending on the task demands.

Top-down comparison of hierarchical representations may depend on identifying corresponding elements in structural descriptions. I will argue that structure-mapping theory (Gentner, 1983, 1989) provides a parsimonious explanation of how people align representations during visual comparison.

I begin by presenting psychological evidence for hybrid and for hierarchical representations. I then discuss structure-mapping as a mechanism for visual comparison over HHRs. Afterwards, I apply HHRs to mental rotation, a commonly studied comparison task. Finally, I consider how individuals strategically search through HHRs during comparison.

2.1 Hybrid Representations

Evidence for hybrid representations comes from at least three bodies of research: categorical perception, object recognition, and visual priming. I consider each of these in turn. Many of the experiments below use the same/different paradigm. In visual same/different experiments (Farell, 1985), participants are shown two images, either sequentially or simultaneously, and asked whether they are the same or different. By varying the type and degree of differences, researchers can study the form of visual representations. Furthermore, by varying the exposure time and the interval between stimuli, researchers can potentially isolate the contributions of encoding, memory, and comparison processes. As we shall see, same/different experiments provide evidence for at least two distinct representations: qualitative and quantitative.

Categorical Perception

Categorical perception is our ability to use qualitative categories when distinguishing between quantitative values. In categorical perception experiments, participants compare stimuli that typically vary along a single dimension (e.g., color: Bornstein & Korda, 1984; size: Kosslyn et al., 1977; or location: Maki, 1982). A common finding is that the more distant the stimuli are along that dimension, the more quickly an individual can recognize that they are different, or that one is greater than the other. However, when the stimuli have different categorical values, performance improves significantly.

For example, Bornstein and Korda (1984) showed participants color patches in a sequential same/different task. They used seven patches that varied from blue to green, such that the middle patch (4) was ambiguous in color. Thus, they could independently vary the quantitative difference and the qualitative difference (same-color vs. different-color) between patches. They found that for pairs with a distance of two, participants responded “different” faster when the patches were different colors than when they were the same color, suggesting that participants used the color labels when comparing. However, participants could also distinguish between different patches with the same label, indicating they had access to the quantitative hue value. Bornstein and Korda theorized that participants were comparing the quantitative and qualitative color values in parallel, and responding as soon as either comparison returned a difference.

Categorical perception has been studied extensively in the color domain (e.g., Bornstein & Korda, 1984; Roberson, Pak, & Hanley, 2008; Regier & Kay, 2009). However, it has also been found in other, more spatial domains. Kosslyn et al. (1977) taught participants to draw 6 stickmen of varying sizes and also trained them to characterize the three largest as “large” and the three smallest as “small.” Similarly, Maki (1982) taught participants the locations of twelve cities on a map, in which six cities were in a northern state and six were in a southern state. Afterwards, participants were given the names of stimulus pairs and asked to judge which was larger or farther north. Both researchers found that participants responded faster when the stimuli were farther apart along the relevant dimension. However, participants also responded faster to stimuli on opposite sides of the category boundary they had learned.

Categorical perception has also been found for spatial relations between an object’s parts. Rosielle and Cooper (2001) used line-drawings of 3-D objects in a sequential same/different study. Each object had two parts connected by an arm. The arm’s orientation varied along four values: 0°, 30°, 60°, and 90°. For the first value (0°), participants might recognize a qualitative “parallel” relationship between the parts. For the last value (90°), participants might recognize a “perpendicular” relationship. However, either of the middle two values would likely be labeled as “oblique.” Rosielle and Cooper compared performance across the conditions where the two objects had identical parts but their angles differed by 30°. They found that participants responded slowest (and error rates were highest) when the object changed from one oblique angle to the other. Participants performed better when the object changed to or from a parallel or perpendicular angle, suggesting they were encoding and using these qualitative spatial relations.

Finally, Ferguson, Aminoff, and Gentner (1996) showed that categorical perception can play a role even when perceiving a single object. They showed participants 2-D polygons and asked them to evaluate whether each polygon possessed an axis of symmetry. This is essentially a same-different task in which the participant is comparing one half of a shape to the other. The experimenters looked at the time and accuracy to detect purely quantitative asymmetries vs. qualitative asymmetries, where the latter included changes in the number of vertices and changes between concave and convex vertices. The experimenters attempted to keep the degree of quantitative asymmetry constant while varying the presence of qualitative asymmetries. They found that participants were better able to detect qualitative asymmetries, suggesting again that participants use qualitative spatial relations during comparison.

These studies indicate that participants consider both quantitative and qualitative features during visual comparison. Features may be purely visual (i.e., color) or spatial. Spatial features may include attributes of individual objects or relations between elements (i.e., relative orientation, convex vs.

concave). It is less clear whether individuals prefer one type or the other. Some researchers (Bornstein & Korda, 1984; Kosslyn et al., 1977) have suggested that qualitative and quantitative features are compared in parallel, so that a large difference along either dimension produces a fast response. We shall return to this question later.

Object Recognition

A heated debate in the field of object recognition provides further evidence. Here, the question is how people rapidly recognize objects from different viewpoints. Biederman and colleagues (Biederman, 1987; Hummel & Biederman, 1992) have proposed that people represent an object as a set of shape primitives, called *geons*, with qualitative spatial relations between the geons. Geons and relations are identified based on *non-accidental properties*, properties of the image that are unlikely to be tied to a specific viewpoint. For example, if two edges along a shape's surface appear parallel from one viewpoint, they will also appear parallel from most other viewpoints. Thus, a given object should produce a similar geon representation from many different viewpoints, facilitating the recognition process.

A set of geons, features of the geons, and spatial relations between geons, make a *geon structural description* (GSD). This approach aligns with the qualitative representations described above in that GSDs capture qualitative features of, and relations between, the parts of an object. It also provides some pointers to what qualitative features might be encoded—those features which are unlikely to have occurred accidentally, due to viewing a scene from a particular viewpoint.

Biederman and colleagues predict that any rotation which does not alter an object's GSD should not interfere with one's ability to recognize it. They have shown (Biederman & Gerhardstein, 1993) that participants can quickly recognize two objects as the same, regardless of rotation in depth, provided they generate the same GSD. Similarly (Biederman & Bar, 1999), participants can quickly recognize that two objects are different, regardless of rotation in depth, when they generate different GSDs, i.e., they differ in a non-accidental property. However, when two objects differ only in a quantitative property, recognizing they are different over rotations is difficult. This fits well with the above finding (Rosielle & Cooper, 2001) that participants can better tell two objects are different when the change in their angles is qualitative, rather than quantitative.

However, these results have not gone unchallenged. Tarr and colleagues (Tarr et al., 1997; Tarr et al., 1998; Hayward & Tarr, 2000) have argued that we use viewpoint-*dependent* representations to recognize objects. They claim that we deal with multiple viewpoints by learning several different representations for a given object, and by mentally transforming between a novel view and the closest known view. Thus, they predict that when an object is unfamiliar, there should be a cost for recognizing it from a novel viewpoint. In support of this, their studies have shown a cost for recognizing objects from new viewpoints, even when the GSDs are mostly identical. These studies used the same paradigms (Tarr et al., 1997) and sometimes even the same stimuli (Tarr et al., 1998) as Biederman and colleagues. Thus, apparently very subtle differences can shift participants between representations that are more or less viewpoint-specific.

Biederman and Bar (1999) suggest one factor is the accessibility of the geons in a 3-D shape. Even differences to a single vertex can make it harder to recognize a rotated version of an object (Biederman & Bar, 1998). Of course, this means the viewpoint-invariant representation is quite brittle. If it is so easily disrupted, then one cannot expect two viewpoints of the same object to produce identical representations consistently. On the other hand, viewpoint-invariant recognition is achievable in some situations (Biederman & Gerhardstein, 1993). Thus, the results suggest: a) we can access a viewpoint-invariant representation, such as GSD; b) this representation can be used to quickly compare two object images; c) when this representation is insufficient, we must fall back on viewpoint-specific information to perform the comparison. Here, again, we see a hybrid perceptual representation. One component is qualitative and primarily viewpoint-invariant. The other contains quantitative and viewpoint-specific information.

Visual Priming

Further support for hybrid representations comes from visual priming studies. Hummel and colleagues (Stankiewicz, Hummel, & Cooper, 1998; Stankiewicz & Hummel, 2002; Thoma, Hummel, & Davidoff, 2004) have argued that a viewpoint-invariant representation such as GSD can only be generated when one attends to an image. Attention allows one to identify the parts of an object and bind them to features and relations (Treisman & Gelade, 1980), thus generating a structural description. If an individual is exposed to an image but fails to attend to it, they will generate a more holistic, viewpoint-specific representation.

In their experiments, two objects are displayed on the screen together. Participants are cued to attend to one, while they are given very little time to notice the other—typically, the two objects together are displayed for only 120 ms. Afterwards, participants are shown another object and asked to name it. The experimenters predict that if an individual attends to an object, they will generate a viewpoint-invariant representation that will prime them for naming that object again, even if the object is transformed. However, if an individual doesn't attend to an object, they will generate only a viewpoint-specific representation. This representation will support priming across a much narrower set of transformations.

Hummel and colleagues found that when participants attended to an object, they were primed to recognize it again, even if the original object was mirror-reflected (Stankiewicz, Hummel, & Cooper, 1998), or scrambled (Thoma, Hummel, & Davidoff, 2004) by splitting the image into two halves and inverting them. Importantly, the priming for both mirror-reflected and scrambled images was greater than the priming for seeing a different object from the same class (e.g., two different pianos), so this was not merely semantic priming.

In contrast, an ignored image failed to prime mirror-reflected or unscrambled versions of itself, although it did prime an identical version, as well as larger or smaller versions (Stankiewicz & Hummel, 2002). Curiously, a scrambled, ignored image failed to prime even an identical scrambled image (Thoma, Hummel, & Davidoff, 2004). This, along with the size invariance, suggests that the ignored image does not simply produce a pixel-by-pixel representation. Some amount of abstraction or recognition appears to occur even for ignored images, although it is insufficient for supporting transformations such as mirror reflection.

Characterizing Qualitative Representations

We have now seen strong evidence for a qualitative/quantitative distinction in visual representations, as well as reasonably strong evidence for a viewpoint-invariant/viewpoint-specific distinction. However, two questions remain: 1) Are the qualitative and the quantitative simply two sets of features in a single representation, or are there real differences in the ways they are represented and reasoned over? 2) How closely tied together are the qualitative representation and the viewpoint-invariant representation? To better answer these questions, I will now make some claims about the nature of qualitative representations.

1) Qualitative representations appear linked to symbolic processing in the brain

Qualitative features (e.g., “blue,” “above,” or “parallel”) often correspond to single words, whereas quantitative features (e.g., “3 inches to the right”) often do not. Therefore, if qualitative and quantitative information are represented separately in the brain, one might suppose that the left hemisphere, known for language-processing (at least in right-handed individuals), would handle qualitative features better. In contrast, the right hemisphere, known for handling active spatial tasks such as navigation, might handle quantitative features better (Jager & Postma, 2003). A number of studies have confirmed that this is the case.

Kosslyn et al. (1989) gave participants a binary forced choice task in which they had to either judge a distance or a qualitative relation between objects. The qualitative relations included topological (on/off) and relative position (right/left, above/below). When stimuli were presented to the individual’s right visual field, meaning they’d be processed in the brain’s left hemisphere, qualitative relation judgments were faster. When stimuli were presented to the left visual field, distance judgments were faster. Thus, participants seemed to process qualitative relations faster in the side of the brain known for processing language. Several follow-up studies have mostly confirmed this phenomenon in the domain of spatial relations (see Jager & Postma, 2003, for a review).

Similarly results have been found in categorical processing of color. Given a set of colors, individuals can identify the odd color out more easily when a) it possesses a different verbal label and b) it is displayed in the right visual field (Gilbert et al., 2006). Follow-up studies (Roberson, Pak, & Hanley, 2008) have suggested that this result depends on how quickly participants respond. If they respond more slowly, categorical perception spreads to both visual fields, just as one might expect information to spread between the hemispheres of the brain.

Hemispheric effects also appear in memory for locations. Huttenlocher, Hedges, and Ducan (1991) found that when participants are shown a dot in a circle and later asked to reproduce the dot's location, their response is biased towards the center of whichever quadrant of the circle contained the dot. This suggests participants qualitatively encoded the dot's quadrant and used that information to aid in remembering the dot's location. Perhaps unsurprisingly, Laeng, Peters, and McCabe (1998) found that participants showed more bias towards the quadrant of the circle when the location had been displayed to their right visual field.

If qualitative representations are handled by the language part of the brain and quantitative representations are handled by the visuo-spatial part, then a verbal interference task should weaken the effect of qualitative representations, while a spatial interference task should strengthen it. This appears to be the case. Gilbert et al. (2006) found that when participants were simultaneously performing a verbal interference task, which made it difficult for them to think linguistically, categorical perception of color vanished. In contrast, Huttenlocher, Hedges, and Duncan (1991) found that when participants were simultaneously performing a spatial interference task, in which they had to remember a spatial array of values, participants showed more bias towards the appropriate quadrant of the circle.

If qualitative representations are connected to language, do they vary from culture to culture, depending on the terms available in a culture's language? This appears to vary by domain. Several studies (Özgen, 2004; Roberson et al., 2005; Roberson, Pak, & Hanley, 2008) have shown that categorical color perception depends on how a culture's language divides up the color space. However, qualitative representations of space may be more constant across cultures. Dehaene et al. (2006) evaluated 2-D spatial knowledge in a South American indigenous group who lacked words for most geometric concepts. Their results indicate the group could utilize such qualitative concepts as parallel, perpendicular, containment, intersection, and symmetry.

2) Qualitative representations are structural

We might infer that just as language is structural, containing syntactic relations between words, qualitative representations are also structural. In support of this theory, people encode qualitative spatial relations, both between objects (Kosslyn et al., 1989) and between object parts (Rosielle & Cooper, 2001). If qualitative representations contain multiple elements (objects or object parts) and relations between those elements, they may be considered full structural representations. Biederman's (1987; Hummel & Biederman, 2002) geon structural descriptions are one possible structural representation for object shape. Two other prominent vision scientists (Palmer, 1977; Marr & Nishihara, 1978) have suggested that we use structural representations for object shape, although they did not focus on qualitative features. Other researchers (Markman & Gentner, 1996; Lovett, et al., 2009a) have suggested that we represent visual scenes structurally, and that we compare two images by aligning their common relational structure.

3) Qualitative representations are easy to remember

Qualitative representations are a form of abstraction—they take a continuously varying dimension and divide it into discrete chunks. This implies it will be easier to remember qualitative values than quantitative values. For example, Huttenlocher, Hedges, and Ducan (1991) argued that participants used circle quadrants in reproducing a dot's location because the qualitative quadrants were easier to remember than the dot's quantitative location. Similarly, it should be easier to remember that an object was red than to remember its exact hue.

Recall the Rosielle and Cooper (2001) study, in which participants compared objects with two parts at different angles to each other. They conducted a second study in which participants were instructed to ignore the angle between the parts, and respond "same" if the two parts were the same. With this variation, qualitative differences failed to play a role; accuracies and reaction times depended only on the quantitative difference between the angles. However, when the time between stimuli increased from 756 ms to 2,016 ms, participants again used the qualitative differences. The experimenters suggested that participants failed to keep the quantitative information in memory for this longer period, and so they were forced to rely on the qualitative relations.

Stankiewicz, Hummel, and Cooper's (1998) visual priming study also suggested that viewpoint-invariant representations are remembered better than viewpoint-specific ones. In their initial experiment, they found that the effects of viewpoint and attention were additive. That is, while individuals who attended to an object were primed to recognize a reflection of that object, they showed additional priming when recognizing the identical object. This suggests that individuals used both viewpoint-specific and viewpoint-invariant information when recognizing objects. However, when the experimenters increased the delay between the prime and probe images from 3 seconds to around 5 minutes (by placing the prime and probe in different blocks), the viewpoint-specific effect went away. When participants attended to the object, they achieving identical priming for the same and mirror-reflected objects, while when they did not attend, they achieved no priming at all. Thus, both qualitative and viewpoint-invariant representations are apparently easier to remember.

4) Qualitative representations are mostly invariant

An advantage of many qualitative features is that they are invariant over transformations to an image. For example, the exact color of an apple may vary depending on lighting, but we will almost always recognize the apple's color label, e.g., "red." The same holds for spatial relations. Suppose two edges are parallel, or one object is touching another. If we rotate the image in space, these relations will almost always remain constant, as long as we can still see their elements. In contrast, the exact location, orientation, or size of an object will change as we see it from different viewing angles. This is why Biederman's (1987; Hummel & Biederman, 1992) geon structural descriptions are based on non-accidental properties, qualitative features that remain constant across most rotations.

Importantly, not all qualitative relations are invariant. Kosslyn et al. (1989) demonstrated that we process the right/left distinction better with the left hemisphere, which processes other qualitative features. However, right/left is easily disrupted by transformations. In particular, it is reversed by mirror reflections. How can we reconcile this with Stankiewicz, Hummel, and Cooper's (1998) finding that mirror reflections do not disrupt image priming?

There are at least three possibilities: 1) Perhaps qualitative representations contain many invariant relations but a few orientation-specific ones, such as right/left. Mirror-reflection priming may work because all the other relations outweigh the small contribution made by right/left.

2) Perhaps there is a distinction between qualitative relations that are automatically encoded and those that we learn over time. Trickier relations like right/left may not be encoded during the brief stimulus exposures used by Stankiewicz, Hummel, and Coopers (1998). In support of this hypothesis, American children often have more difficulty learning the right/left distinction than learning other distinctions like above/below (Corballis & Beale, 1976; Harris, 1972), and difficulties with right/left can persist into adulthood (Maki & Marek, 1997), although the difficulties may be more linguistic than perceptual by adulthood (Sholl & Egeth, 1981). Furthermore, some cultures do not even have words for "right" and "left" (Levinson, 1996).

3) Perhaps right/left is commonly encoded between objects but rarely encoded between parts of an object, since it is less useful for supporting orientation-invariant recognition. This is relevant because Stankiewicz and colleagues used single objects as stimuli. Note that explanations 2) and 3) are consistent.

5) Qualitative representations are brittle

Unfortunately, qualitative representations can be quite brittle. Suppose we have two color patches, both of them about halfway between yellow and green. How are we to label them? Perhaps one is barely closer to yellow, and so it gets labeled “yellow,” while the other gets labeled “green.” If one only compares the qualitative representations, these patches appear to be entirely different. However, if one considers their quantitative values, they are actually quite similar.

Biederman and Bar (1998, 1999) encountered this same problem with viewpoint-invariant recognition. When an object was rotated in space, very subtle differences, even in a single vertex of one surface, resulted in participants being less likely to identify the objects as the same shape. Geon structural descriptions predict this because subtle differences might result in a different geon type being encoded, resulting in an entirely different structural description.

There is always a chance that two similar values will receive different qualitative labels because they happen to lie on either side of a threshold. In addition, two different values may receive the same qualitative label because they lie on either extreme of a category. Thus, we cannot depend solely on qualitative representations. Instead, *quantitative* information may be needed to support or contradict the results of qualitative comparisons.

Qualitative and quantitative representations

Qualitative representations appear to be distinct from quantitative representations. I have argued that they are symbolic, structural, sufficiently abstract to be remembered easily, and often invariant over transformations. In these respects, they are excellent for image comparison, especially when an image must be remembered over time. Unfortunately, they are not always reliable. In addition, they do not always provide a sufficient degree of precision: sometimes, different images share the same qualitative features. Thus, they must often be supplemented by quantitative representations.

Unfortunately, the form of quantitative representations is less clear. Even Tarr and colleagues (Tarr et al., 1997; Tarr et al., 1998; Hayward & Tarr, 2000), who argued that we use quantitative representations for object recognition, were rarely clear about what those representations look like. I believe this is an important area for further study.

2.2 Hierarchical Representations

On their own, hybrid representations are not sufficiently flexible to support visual comparison. For example, the studies above refer to representations of both objects and visual scenes. How do these representations relate? And how does an individual know, when presented with a novel image, whether to represent it as an object or as a scene? To answer these questions, I appeal to hierarchical perceptual representations.

Several researchers (Marr & Nishihara, 1978; Palmer, 1977; Hochstein & Ahissar, 2002) have argued that perceptual representations are hierarchical. Marr and Palmer each presented models in which an object is represented structurally, as a set of elements with attributes for each element and relations between elements. Importantly, each element can be decomposed into a set of sub-elements with their own structural representation. For example (Marr, 1976), a person is a single object. However, a person is also a torso connected to two arms, two legs, and a head. An arm is a fore-arm and an upper arm. A fore-arm is an arm, a wrist, and the hand. And the hierarchy can continue as needed.

Both Marr and Palmer argued that hierarchies support flexible comparison because we can begin by comparing the highest level and then move down as needed. This is consistent with Hochstein and Ahissar's (2002) *reverse hierarchy theory* of perception. According to reverse hierarchy theory, perceptual representations are constructed bottom up, moving from local features to objects to the overall gist of a scene. However, only the top level is consciously available to an individual. We must apply attention and effort to move back down the hierarchy and recognize the smaller-scale elements in a scene. Thus, a brief fixation (e.g., 100 ms) is often enough to determine what is happening in an image (Biederman, 1974; Intraub, 1999), but more time is required to identify the objects and relations.

Treisman and Gelade's (1980) feature-integration theory may help explain the underlying mechanisms. They argued that pre-attentive perception is not very good at binding features to individual objects. For example, if we see a red square next to a blue circle, we may not initially recognize that the square is the red shape and the circle is the blue shape. Only by applying attention to the objects can we determine with certainty which features go together in each object. If this is true, then binding objects to relations in a structural representation is likely even harder. Thus, attention must be applied to construct a structural representation, as in Hummel and colleagues' visual priming experiments (Stankiewicz, Hummel, & Cooper, 1998; Stankiewicz & Hummel, 2002; Thoma, Hummel, & Davidoff, 2004).

The model of perception that emerges is something like the following: when we first view an image, bottom-up processes group local elements together and attempt to determine, from the jumble of features, what is going on in the scene. They generate a best guess at a high level of abstraction, perhaps treating the entire scene as one or a few objects. Then, an individual can improve precision and correct possible mistakes by attending to the individual elements, thereby constructing detailed, structural representations for each element's components.

This model predicts that during visual comparison, people will consider high-level structure before low-level structure. There are two reasons to think this. Firstly, high-level structure is more abstract and sparser. Thus, the comparison is simpler. For example, in Figure 2.1A, if we compare the groups of circles, we can easily determine that most images contain a single group, while one contains a group with a solo circle. However, if we compare the individual circles, the comparison is more difficult—we must consider the distances between each pair of circles. In this case, a high-level comparison is sufficient. In cases where the results are unclear, we can break down the images into their elements and perform a more fine-grained comparison (Palmer, 1977).

Secondly, high-level structure is available first. We must apply attention to produce low-level structure. Therefore, we can compare higher-level structure before lower-level structure has been computed. This is especially useful when there is limited time available for encoding.

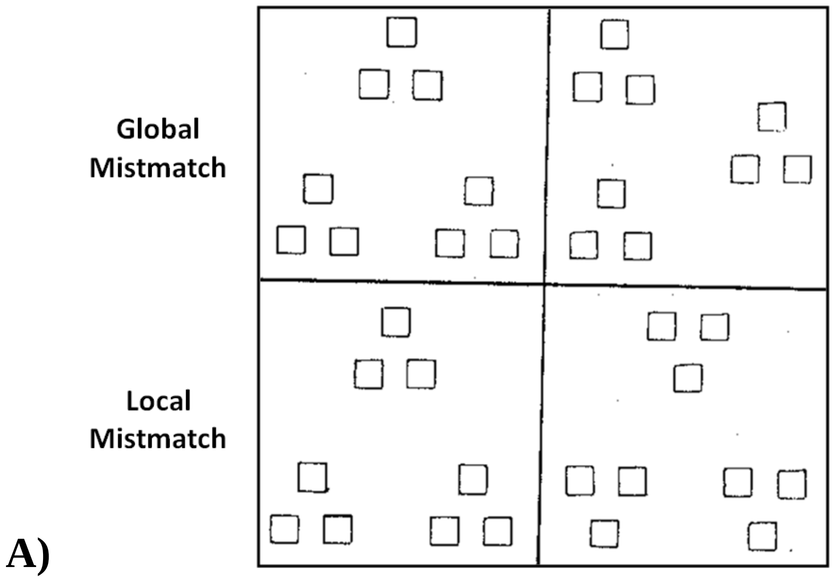
I now review evidence supporting the prediction that high-level structure is compared before low-level structure. Next, I consider two further implications that remain more theoretical.

Psychological Evidence

Navon (1977) pitted high-level structure against low-level structure in a same-different task. His stimuli consisted of 9 squares (Figure 2.2A). Sets of three squares were grouped into triangles, and then three such triangles were grouped into larger triangles. The small-scale triangles (from three squares) and the larger-scale triangles (from three groups) each had an orientation, which could vary independently.

When participants compared two stimuli, there were three possibilities: the stimuli were identical, their large-scale triangles differed (global mismatch), or their small-scale triangles differed (local mismatch). Navon (1977) performed two experiments: a sequential same-different with limited time to encode the first stimuli, and a simultaneous same-different with limited time to encode and compare both. In each case, participants responded "different" more accurately when the large-scale triangles differed than when the small-scale triangles differed, suggesting that the large-scale triangles were available first. Importantly, Navon conducted controls in which the sizes of the images varied, as well as controls with only a single small-scale triangle. These results confirmed that participants had no difficulty with the smaller sizes of the small-scale triangles. Rather, participants appeared to prefer representations at the largest scale available for a given image.

Love, Rouder, and Wisniewski (1999) conducted same-different studies with arrays of nine shapes (Figure 2.2B). Again, the shapes could be grouped into larger configurations. However, here the grouping was based on similarity, rather than proximity. Of the nine shapes, three were circles, three were squares, and three were triangles. Thus, shapes could only be grouped in a bottom-up fashion, based on the local similarity of the shapes. This rules out one potential confound in the above study: participants could not perceive the groupings by focusing on lower spatial frequencies because high spatial frequencies were required to distinguish the three shape types.



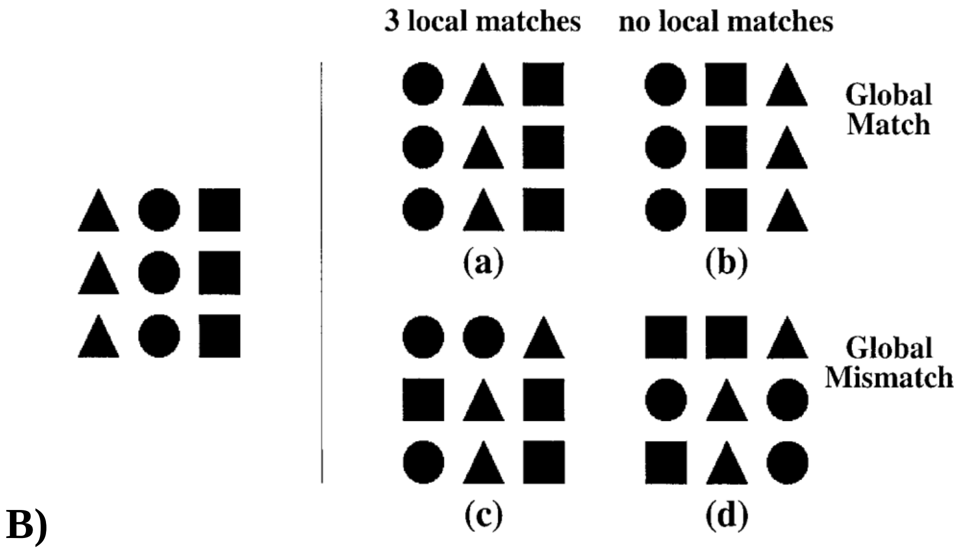


Figure 2.2 Stimuli from (A) Navon (1977, Experiment 4), and (B) Love, Rouder, and Wisniewski (1999) (B).

When pairs of stimuli were different, they could be locally similar/dissimilar and globally similar/dissimilar. Locally similar meant that three of the nine shapes were the same, while locally dissimilar meant there were different at every location. Globally similar meant the groupings of similar shapes were identical, while globally dissimilar meant they were not. Overall, Love and colleagues found that both types of similarity affected performance: participants responded “different” faster when stimuli were either locally or globally dissimilar. However, there was an interaction between the similarity types: when stimuli were globally similar, local dissimilarity speeded performance to a large degree (104 ms). When stimuli were globally dissimilar, local dissimilarity contributed significantly less (21 ms). This suggests that participants typically compared global configurations first. When these were dissimilar, they could respond "different" quickly, and so local similarity mattered little. When they were similar, participants had to consider the individual objects, and so local similarity mattered far more.

Further Implications

We can draw two additional implications from hierarchical representations. I consider each of these in turn.

1) Representations of shape are like representations of space

The studies above have mostly looked at either the shapes of individual objects or the spatial layouts of visual scenes. However, if humans can generate hierarchical representations of arbitrary depth, then there may be no strong distinction between shape representations and spatial representations. We may utilize hybrid perceptual representations, as described above, for a scene with multiple objects, an object with multiple parts, or a part with multiple edges. At any level, we may identify a set of qualitative spatial relations between elements to complement a more exact quantitative representation.

While there is no direct evidence for this hypothesis, we have seen that people can utilize qualitative spatial relations either between objects (Kosslyn et al., 1989; Huttenlocher, Hedges, & Duncan, 1991) or between parts of an object (Rosielle & Cooper, 2002; Ferguson, Aminoff, & Gentner, 1996). This suggests that people could generate qualitative, structural representations at either level.

This does not mean people will identify the exact same qualitative relations at each level in the hierarchy. I have already suggested that the right/left distinction may be more useful between objects than between object parts. Some relations, such as a convex corner between two edges (Ferguson, Aminoff, & Gentner, 1996) are almost meaningless at other levels of abstraction—while it might be possible to identify a convex corner between two objects, this is surely not a relation people normally consider. Thus, the *content* of representations will surely change between levels. However, the *format* may remain relatively constant.

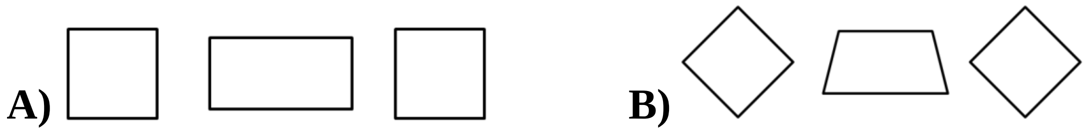


Figure 2.3 Two images that one might compare.

2) High-level comparisons can guide low-level comparisons

I have suggested that people first perform high-level comparisons and then scale down as needed. However, there are two possible ways to move down the hierarchy (Marr & Nishihara, 1978). One is to break down the elements into smaller components but still represent the entire image. This provides a more detailed representation of the visual scene. However, this representation could quickly become unwieldy. For example, imagine representing all the individual edges in the objects of Figure 2.3. There would be 24 total edges, and a huge number of relations.

A second approach is for high-level comparison to guide the low-level comparisons. Consider again Figure 2.3. A comparison between structural representations at the object level could identify the corresponding object in the two images. For example, the rectangle in image A goes with the trapezoid in image B because they occupy the same spot in the relational structure. Once the corresponding objects are identified, one can compare the edge-level representations for each object pair. Thus, instead of comparing two images with 24 total edges, we're comparing two objects with four edges each.

By comparing the qualitative edge-level representations, one can determine that the left object (the rectangle) has more parallel edges than the right object. This might be sufficient for the current comparison. However, one could take the comparison a step farther. The structural comparison at the edge level should provide the corresponding edges. One could now compare each corresponding edge pair. At this level, there is very little structural information. However, by quantitative length values, one could determine that the topmost edge has become shorter.

2.3 Structural Comparison

I have argued that we perceive a hierarchy of hybrid representations. At each level in the hierarchy, there is a qualitative, structural representation, as well as a complementary quantitative representation. These representations require a structural comparison process to align two representations and determine their commonalities and differences. The process should also identify corresponding elements in the representations, to guide lower-level comparisons. Finally, the process should be general enough to handle different content at different levels in the hierarchy, provided that the format remains constant.

I propose that structure-mapping theory (Gentner, 1983; Gentner, 1989; Falkenhainer, Forbus, & Gentner, 1989) provides such a process. According to structure-mapping theory, people compare structural representations by aligning their common relational structure. Representations consist of entities, attributes of entities, and relations between entities. There are also higher-order relations between other, lower-order relations. Comparison is biased to prefer aligning deep structure (i.e., higher-order relations) over shallow structure. For example, consider an analogy between soccer and basketball. In this analogy, the colors of the balls (white vs. orange) fail to align. However, the higher-order causal structure (moving a ball through a net results in scoring points) aligns much better. This deep structural alignment is preferred, and thus there is a good analogy between these sports.

Structure-mapping results in a *global mapping* between two representations. A global mapping consists of a) *correspondences* between the entities, attributes, and relations; b) a similarity estimate based on the depth and breadth of aligned structure (with depth preferred); and c) *candidate inferences* based on elements in one representation that failed to align with the other—for example, if one knows being tall helps score points in basketball, one might infer that being tall will also help in soccer, based on the analogy.

Global mappings thus provide several useful pieces of information. The corresponding entities identify which objects go together in the two representations, e.g., the soccer ball maps to the basketball. The corresponding attributes and relations identify what is common to both representations, e.g., both involve scoring with nets. The candidate inferences identify differences between the representations, e.g., the colors of the balls. Finally, structure-mapping provides a similarity measure.

Structure-mapping was originally proposed as a model of abstract analogy in people. However, there has been increasing evidence that we use a similar process in concrete visual comparisons (Markman & Gentner, 1996; Lovett, et al., 2009a; Sagi, Gentner, & Lovett, in press). If qualitative perceptual representations are structural, then structure-mapping provides an appropriate and useful process for comparing them.

2.4 Mental Rotation: An Example Domain

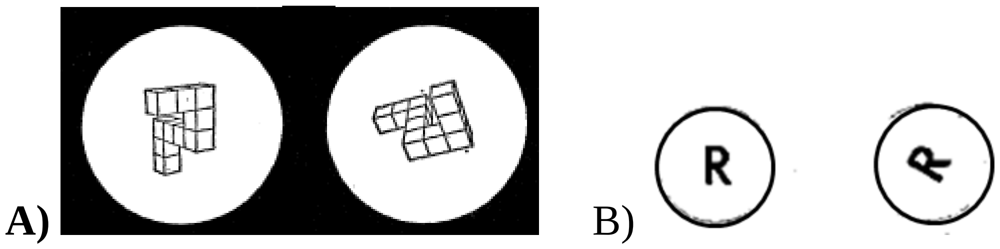


Figure 2.4 Mental rotation stimuli from (A) Shepard and Metzler (1971), and (B) Cooper and Shepard (1973).

I now apply the HHR model to mental rotation, an extensively studied comparison task. In the classic mental rotation paradigm (Shepard & Metzler, 1971), participants are shown two objects and asked whether a rotation of one object would produce the other. The stimuli can be three-dimensional block figures (Figure 2.4A) or two-dimensional shapes, such as letters (Figure 2.4B). Often, but not always, distractors are mirror-reflections. Thus, even when there is no valid rotation between the shapes, their internal structure can be quite similar.

Consistently across studies, the time to say two objects are rotations is proportional to the degrees of rotations between them (Shepard & Cooper, 1982). For example, participants will respond faster if objects are 45° apart than if they are 90° apart. This has led to the hypothesis (Shepard & Metzler, 1971) that people perform an analog rotation in their minds, transforming one object's representation to align it with the other's. Such a process would likely operate on quantitative, orientation-specific representations. Once these representations have been aligned, one can easily compare them.

Mental rotation can be seen as a special case of object recognition. The key difference is that in recognition, a mirror-reflected object is considered the same, whereas in mental rotation, a mirror-reflected object is considered different. This may be why there is a cost for displaying objects at different orientations in mental rotation, but there is often little cost in recognition (Biederman & Gerhardstein, 1993; Biederman & Bar, 1999): if qualitative shape representations are invariant over reflection (Hummel & Biederman, 1992; Stankiewicz, Hummel, & Cooper 1998), they should be sufficient for recognition but not for mental rotation.

A mystery remains in the mental rotation results, however. Two-dimensional objects can be rotated in two directions: clockwise and counter-clockwise. Three-dimensional objects can be rotated along many different axes. Despite this, reaction times are usually proportionate to the angle along the shortest axis of rotation, in the closest direction (Shepard & Cooper, 1982), unless the axis is an unfamiliar one (Parsons, 1995). How do people know which way to rotate the object before they know whether the objects align? And how do they know how far to rotate?

Shepard and Cooper (1982) suggested that before rotating, people identify corresponding parts in the two shapes. These corresponding parts could then guide the rotation. But how could we find corresponding parts in objects at different orientations? I believe the key is structural comparison over qualitative, orientation-invariant representations. I propose a three-stage model for mental rotation, based upon HHRs:

1) Encode a qualitative, orientation-invariant representation for each object. Compare the representations via structure-mapping to identify corresponding parts. For example, consider the letters in Figure 2.4B. Their representations might be quite simple—a long straight edge, a short straight edge, and a curve are all connected. A comparison should easily determine the mapping between edges.

2) Determine the axis and angle of rotation between a pair of corresponding parts. For example, one might consider the long edge in the two shapes. Because these are single, rigid parts, finding a rotation between them is simple. One should immediately see that there is a 60° rotation, if we rotate counter-clockwise about this edge's center.

3) Apply this rotation to the quantitative, orientation-specific representation of the first object. Because all the parts are being rotated together, this rotation is more effortful. Thus, there is a cost for the degree of rotation. Once one representation is rotated, the two representations can be quickly compared for identicality.

This model makes a few assumptions about qualitative and quantitative representations. Firstly, the qualitative representations must be compared before the quantitative ones. As we've seen above, it is sometimes unclear whether one of these will be compared before the other. However, in this case, the qualitative constrains the quantitative.

Secondly, the elements in the qualitative representation are likely tied to the quantitative, perhaps through *indices* (Pylyshyn, 2001) pointing to an element's location in the quantitative space. This would explain how individuals can compute a rotation in the quantitative representation based on corresponding elements in the qualitative.

Finally, the qualitative representation can be manipulated in an analog manner, as when it is rotated. This was the conclusion of the original researchers on mental rotation (Shepard & Metzler, 1971; Cooper & Shepard, 1973). It has given rise to a large body of work on mental imagery (e.g., Kosslyn, 1980, 2005), our ability to generate and manipulate spatial representations. A key idea of imagery is that a concrete representation can be generated from a more abstract one, again showing the qualitative to quantitative direction of reasoning. Note, however, that imagery remains a controversial topic. Kosslyn's model has been met with repeated objections over the years (e.g., Pylyshyn, 1981, 2002).

I now consider some model variants that people might utilize.

Model Variants

According to the model above, we fully encode each object's qualitative structure before comparing. However, one does not need a complete mapping between the objects—only one corresponding pair of parts is required. Thus, individuals may be lazier, only encoding the bare minimum required to find a corresponding pair. For example, when faced with block diagrams, participants reported trying to line up one end of an object with one end of the other object (Shepard & Metzler, 1971). Thus, their initial comparison only needed to map between the two ends of each object. They might well have ignored the structure in the objects' interiors.

On the other hand, suppose one fully encodes each object's structure for the structural alignment. This initial comparison might then provide significant information about the similarity between the objects. If the alignment between the objects is poor, the individual can immediately determine that the objects are different. However, if the alignment is strong, the individual must perform the quantitative rotation to determine whether the objects are the same.

Suppose that two objects are actually different. There are at least three cases where a qualitative alignment will fail to reveal this difference, thus requiring a quantitative rotation: a) The individual may not have fully encoded the objects at the qualitative level. b) The objects may be mirror reflections. If qualitative, orientation-invariant representations are constant across reflection (Hummel & Biederman, 1992; Stankiewicz, Hummel, & Cooper 1998), then no difference will be detected. c) The objects may have the same qualitative features but vary along a quantitative dimension, such as the exact size or orientation of a part. I now consider studies looking at cases b) and c).

Mirror reflection in mental rotation

If mental rotation depends on a qualitative comparison followed by a quantitative comparison, then we can predict one way individuals might fail: an individual might perform a careful qualitative comparison but skip the quantitative comparison. This should result in a particular pattern of errors. If two objects have qualitatively different shapes, the individual should accurately determine that one is not a rotation of the other, without ever actually rotating. However, if the objects are mirror reflections, the individual should be unable to tell that they are different.

In fact, some individuals show exactly this pattern of errors on a mental rotation standardized test (Geiser, Lehmann, & Eid, 2006; Janssen & Geiser, 2006). These individuals, called "non-rotators" do quite well on problems that don't require quantitative rotation, out-performing other groups such as the "slow rotators." However, they do worse than slow rotators on problems that require rotation, i.e., the problems with mirror-reflected distractors. It should be noted that only a small percentage of participants are non-rotators (13.2% and 7.5%, respectively, in the two studies). This may be because participants capable of generating full qualitative representations are usually able to rotate effectively.

Qualitative vs. quantitative differences in mental rotation

Biederman and Bar (1999) evaluated same-different performance for qualitative vs. quantitative differences. The differences were equally easy to spot when objects were displayed at the same orientation. However, they predicted diverging results when the objects were rotated: quantitative differences should be much harder to spot, while qualitative differences should be about the same. This is consistent with the HHR model because quantitative differences can only be detected after rotating one shape to align it with the other.

We can make a stronger prediction. When two objects are the same, responding "same" should be slower when there is a rotation. This is because the quantitative representations must be rotated to check for quantitative differences (as in most mental rotation studies). However, when the objects are qualitatively different, responding "different" should be equally fast with or without a rotation. This is because the initial qualitative alignment should be sufficient for ruling out a match. Note that this strong prediction will not always hold. As stated above, participants may not fully encode each object's qualitative structure, and thus they may miss differences at the qualitative level. Additionally, since rotations *can* change features in the qualitative representation, detecting qualitative differences doesn't guarantee that the objects are different. However, the prediction might be born out when a) objects are relatively simple, meaning they are easy to fully encode; and b) rotations are designed to minimize changes to the qualitative structure. Both of these are true in the Biederman and Bar (1999) study.

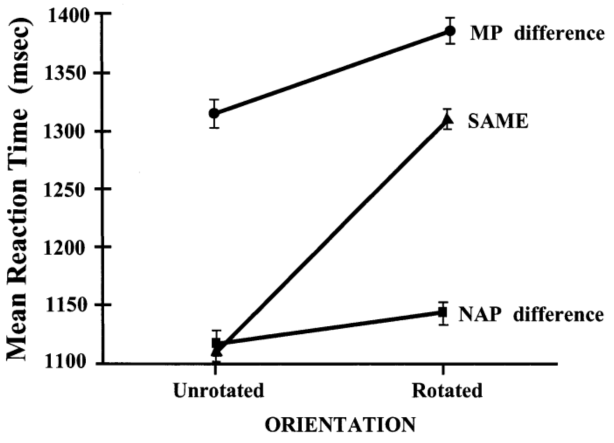


Figure 2.5 Reaction time results from Biederman and Bar (1999, Experiment 2).

Biederman and Bar (1999) ran two conditions between participants: pure block and mixed block. In the pure blocks, participants only saw stimulus pairs that were rotated, or only saw pairs that were not rotated. In the mixed blocks, participants saw both rotated and unrotated pairs. The results were inconclusive for the pure blocks: there was a small cost for rotation on both the "same" and qualitative "different" trials. Perhaps participants always performed the same operation because they knew whether there would be a rotation before each trial began. However, in the mixed blocks, the results were as predicted: there was a cost for rotation on the "same" trials, but no cost on the qualitative "different" trials (Figure 2.5, qualitative differences are NAP, or non-accidental properties). Apparently participants could identify qualitative differences without rotating, but they quantitatively rotated objects before declaring them "same.[1]"

2.5 Strategic Comparison

I have argued that we perform visual comparisons over hierarchical hybrid representations. Representations are hierarchical in that they represent an image at different levels of abstraction, focusing on large-scale groups, individual objects, or parts within each object. They are hybrid in that qualitative and quantitative features are each encoded and can be reasoned over separately. A question remains: how do we determine the *best* representation or representations for a given comparison? This can be thought of as a strategic search through the available representations, aimed at finding ones that maximize informativeness while minimizing cognitive load. Though the details of this process likely vary from task to task, I can propose some general principles.

I believe people move top-down through the perceptual hierarchy (Hochstein & Ahissar, 2002; Navon, 1977). They begin with the sparsest, most abstract representation and incrementally add detail until there is sufficient information. Top-down comparison comes in two forms, which can be labeled *holistic* and *analytic*. In holistic top-down comparison, we encode the entire image at multiple levels of abstraction. For example, consider the oddity task problems in Figure 2.1. In problem A, one would begin at the highest level, representing the images as rows of dots. Here, the highest level is sufficient, as one image contains an additional dot outside the row. If the highest level were insufficient, one might instead consider the spatial relations between the individual circles.

[1] Note the uniformly high cost for quantitative differences ("MP differences") in this condition, suggesting that when participants were unsure whether there'd be a rotation, they grew increasingly dependent on the qualitative features, and it became increasingly difficult to use the quantitative features. Accuracy was quite low when there were quantitative differences only.

In problem B, the highest level is the individual objects. At this level, one could solve the problem if the objects were labeled as “parallelogram” and “trapezoid.” If not, there would be nothing to distinguish the objects at this level, requiring one to consider the edges within each object. At this lower level, the spatial relations are sufficient for performing the task.

On the other hand, in analytic top-down comparison, corresponding elements at one level guide comparisons at the next level down. For example, in Figure 2.3, an in initial comparison would align the rectangle with the trapezoid and the squares with the diamonds. One could then compare the individual edges in each pair of corresponding objects.

The qualitative-quantitative interaction is trickier to ascertain. Some researchers (Kosslyn, et al., 1977; Bornstein & Korda, 1984) have argued that we compare qualitative and quantitative features in parallel. However, these researchers used simple stimuli that varied along a single critical dimension. As stimuli become more complex, the load on memory and cognitive processes increases. In such cases, people may rely more on qualitative representations that are more abstract and easier to remember (Rosielle & Cooper, 2001; Stankiewicz, Hummel, & Cooper, 1998).

In addition, quantitative representations are rigid and viewpoint-specific, while qualitative representations can match across transformations (Biederman, 1987; Biederman & Bar, 1999; Stankiewicz, Hummel, & Cooper, 1998; Thoma, Hummel, & Davidoff, 2004). Thus, qualitative representations can support the identification of systematic differences between images. I explored the example of mental rotation above. I proposed that we compare qualitative representations, identify a transformation, and then transform and compare quantitative representations. Thus, while it is unclear whether qualitative representations are *encoded* first, I believe people will often *compare* them first, especially when dealing with complex stimuli or when seeking systematic differences.

I propose that people perform visual comparison top-down, comparing qualitative features before quantitative at each level, and often using the qualitative correspondences at one level to guide comparisons at the next. However, this is only a general approach. People will vary their strategies depending on the task.

In fact, people may not always use representations rationally. For example, there is some evidence that we attend *more* to quantitative features when we're instructed to ignore them. Bornstein and Korda (1984) compared two same-different tasks: in the *identity* task, participants responded "same" when two color patches were identical, while in the *category* task, they responded "same" when the patches were the same color category, i.e., blue or green. Thus, in the category task, quantitative values were irrelevant. Similarly, Rosielle and Cooper (2001) conducted two versions of their same-different task, where participants compared objects with two parts connected by an angle. In their identity task, objects needed to have identical angles to be considered "same," whereas in the object recognition task, only the parts mattered; the angle was to be ignored. Both experiments found that in the identity task, participants made heavy use of qualitative features in their comparisons. However, in the other task, in which participants were instructed to ignore the exact colors or the angle between parts, participants actually relied *more* on this quantitative value.

Even top-down reasoning can be affected. For example, Navon (1977) showed participants large letters made from smaller letters (Figure 2.6) and asked them to name either the large letter (global condition) or the small letters (local condition). He found that the large letter could interfere with the smaller letters, i.e., people named the smaller letters slower when the large letter was different. In contrast, the smaller letters did not interfere with the large letter. This indicates that people identified the large letter first, consistent with top-down reasoning. However, Lamb and Robertson (1988) changed the task by varying the letter sizes between trials. They showed that when the task context focused participants on smaller-scale letters, the effect reversed, suggesting that participants were now identifying the smaller letters before the larger letters.

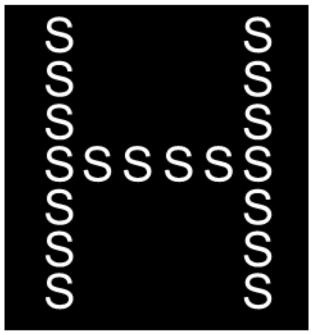


Figure 2.6 Image similar to stimuli from Navon (1977).

These exceptions further illustrate the flexibility of visual encoding and comparison. Such flexibility is absolutely necessary. Even the simplest images contain vast amounts of information. When we compare images, we must search through this information to find the critical commonalities and differences. I believe hierarchical hybrid representations are a means of managing the information. High-level qualitative representations provide an abstract overview of an image. By structurally comparing these representations, we can both recognize qualitative differences and identify corresponding elements in the images. The correspondences then guide the search through the hierarchical space. At any level, qualitative representations can be supplemented with quantitative information. The quantitative representations are not decomposable, but they can be rotated and otherwise manipulated to identify quantitative differences between images. Through strategic use of these representations and processes, we can compare images efficiently across a multitude of scales and visual domains.

3. Modeling Perception and Comparison

Here I describe a computational model of hierarchical hybrid representations. The model is designed for 2-D image comparison. It extends previous work on sketch understanding and structural comparison, described below. Rather than fully implementing HHRs, I make two simplifying assumptions.

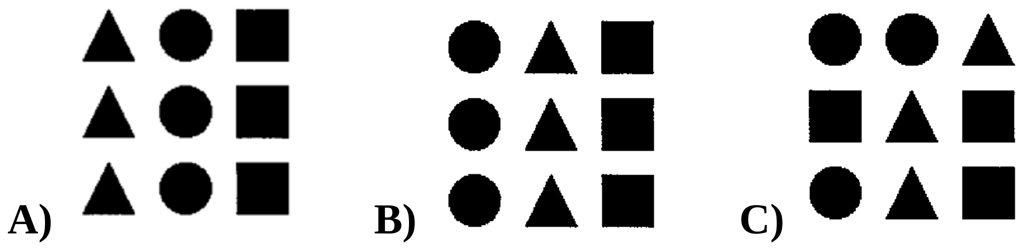


Figure 3.1 Example stimuli from Love, Rouder, and Wisniewski (1999).

Firstly, I use a three-level hierarchy: groups, objects, and edges. The middle level, objects, represents the 2-D objects in an image, e.g., the nine shapes in Figure 3.1A. The top level, groups, represents sets of objects, grouped by similarity and proximity, e.g., the three columns. The bottom level, edges, represents the individual edges in an object, e.g., the three edges in each triangle. I do not claim these are the only levels people represent. They are a starting part for evaluating hierarchical representations in spatial problem-solving.

Secondly, I model image comparison between only *qualitative* representations. Thus, the model will notice a quantitative change, such as an increased distance between objects, only if it results in a qualitative change. This simplifying assumption represents a bet: because qualitative representations are more abstract, better maintained in memory, and easier to compare, people will rely on them whenever possible. The model predicts that problems relying on quantitative image comparison will be much harder for people.

However, the model does use quantitative representations for shape comparison. It can identify transformations between two objects' shapes by considering changes to each edge's length and orientation. Similarly, it can determine if two groups are the same by considering whether the objects remain in the same location. Thus, the model predicts that when people focus on a particular object or group, rather than the entire image, quantitative information will play a role. Indeed, it is necessary for shape transformations (see the previous chapter's model of mental rotation).

I now describe CogSketch and the Structure-Mapping Engine, our pre-existing computational models of sketch understanding and structural comparison. I then present the Perceptual Sketchpad, my CogSketch extension which generates representations in a three-level hierarchy. I describe the vocabulary of qualitative terms for each level. Next, I explain my models of image comparison and shape comparison. Finally, I show how top-down comparison can lead to *perceptual reorganization*, where an image is segmented into a different set of objects to facilitate the comparison.

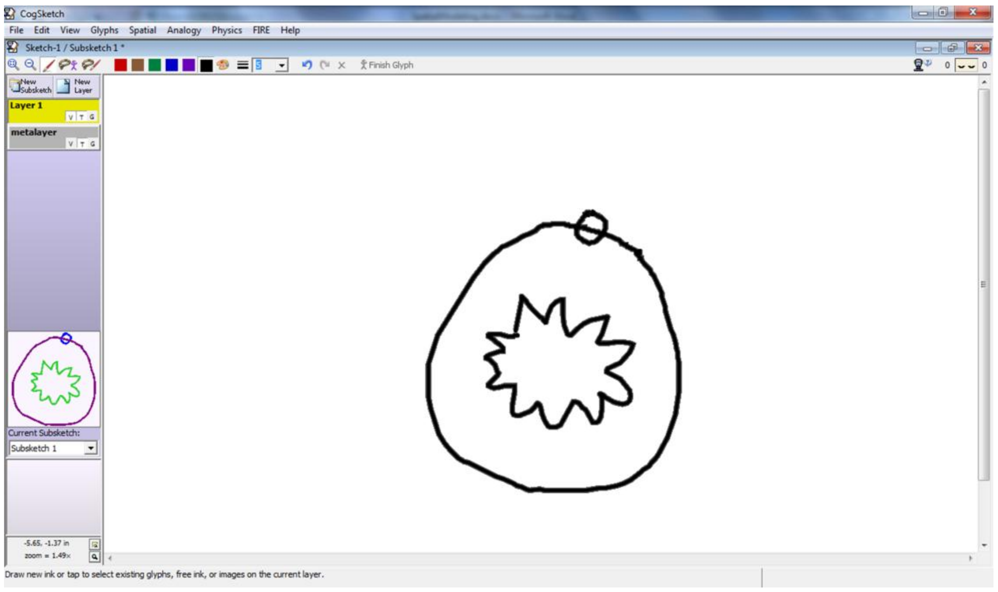
3.1 Existing Models

This work builds on two existing models: CogSketch generates qualitative representations, while the Structure-Mapping Engine compares structural representations. I describe each of these in turn.

CogSketch

CogSketch (Forbus et al., 2011) is an open-domain sketch understanding system. It automatically encodes the qualitative spatial relations between objects in a 2-D sketch. It can work on sketches of nearly any topic, but it requires the user to provide several key pieces of information.

Firstly, the user must segment the sketch into objects. To create a sketch, a user simply begins drawing. CogSketch captures the sketch's *ink*, lists of points describing each stroke drawn by the user. However, the user must press a button to indicate when they have finished drawing one object and begun on the next. For example, consider the sketch in Figure 3.2. This sketch contains three objects: a sun, a planet, and an orbit. The user would draw one object, hit the "Finish Glyph" button, and then begin drawing the next object. Note that CogSketch also provides segmentation tools so that users can segment the ink into separate objects after the fact.



Draw new ink or tap to select existing glyphs, free ink, or images on the current layer.

Figure 3.2 The CogSketch sketch understanding system.

Secondly, the user must indicate what each object represents. This is done by *conceptual labeling*, selecting concepts from a knowledge base to describe the object. For example, the big circle could be labeled "OrbitalPath," while the small circle could be labeled "Planet." These are both concepts in the OpenCyc knowledge base[2]. The knowledge base contains over 58,000 concepts from a broad variety of domains.

Given a sketch, CogSketch automatically computes qualitative spatial relations between objects. These relations include topology (i.e., containment and intersection) and relative position. The relations are a basic model of the features a person might perceive in the sketch. This *perceptual* information combines with the *conceptual* labels to produce a qualitative representation. Using this representation, other systems can compare sketches or reason about how a sketch's objects will interact.

[2] http://www.opencyc.org/

There are two reasons CogSketch is useful for modeling spatial problem-solving. First, it can automatically encode psychological stimuli. Aside from manually sketching, users can also import shapes from PowerPoint. Thus, if stimuli are in, or can be recreated in, PowerPoint, CogSketch can process them. This allows us to run the model on the exact same stimuli as those given to human participants. Second, it can model perception. Conceptual labeling is purely optional. In all the simulations I will describe, I use only perceptual representations. Using CogSketch, I generate representations that align with how people might perceive a visual scene.

Structure-Mapping Engine

The Structure-Mapping Engine (SME) (Falkenhainer, Forbus, & Gentner, 1989; Forbus & Oblinger, 1990) is a computer implementation of structure-mapping theory's comparison operation. It performs structural comparison over symbolic representations, encoded in predicate calculus. Given two cases, a *base* and a *target*, it computes one to three mappings between them. Each mapping contains correspondences between the base and target, candidate inferences based on elements that failed to align, and a similarity score.

Correspondences are computed by aligning expressions with identical predicates. For example, (**rightOf** Object-A Object-B) might align with (**rightOf** Object-1 Object-2). Because these expressions are aligned, SME would also align Object-A with Object-1, and Object-B with Object-2. Alternatively, suppose SME had aligned these objects for other reasons, and that there was no (**rightOf** Object-1 Object-2) relation. Because the **rightOf** relation exists for objects A and B, SME would compute a candidate inference, suggesting that it might also exist for objects 1 and 2.

The similarity score is based on the depth and breadth of aligned structure in a mapping. It is computed by assigning a score to each expression correspondence and allowing scores to trickle down to correspondences between their arguments. First-order relations, which take entities as their arguments, will trickle down one level. Higher-order relations, which take other relational expressions as their arguments, will trickle down multiple levels and receive higher scores. Thus, higher-order relations have more weight in SME, both while computing a mapping and while scoring one.

SME allows constraints to be placed on the mapping. One can require that elements with certain attributes only map to other elements with those same attributes. This is useful for modeling more conservative comparisons. For example, suppose one assigns a unique attribute to each shape type. One can require that objects only map to other objects with the same shape type. I will discuss conservative comparison more in Chapter 4.

The present model makes a few changes to SME's default parameters. Unless otherwise mentioned, these changes are applied universally throughout the model. Firstly, the model makes two changes to increase the optimality of SME's mappings. SME is a heuristic mapper, using a greedy process to quickly combine matches into a mapping. This process can make mistakes, resulting in a suboptimal mapping. The model flips a flag telling SME to use a less greedy merge, increasing the odds that less obvious mappings will be identified. The model also flips a flag telling SME to score matches more conservatively (see Appendix A).

Secondly, the model asks SME for candidate inferences in both directions. Normally, candidate inferences are computed only forward, from the base to the target. This change allows the model to identify differences between two representations, based on expressions in either that failed to align.

Thirdly, the model increases the max score for a single expression match. Normally, scores are capped at 1.0. While this is fine for image and shape comparisons, more abstract comparisons (described in later chapters) involve several levels of structure. The trickle-down across these levels can produce scores far higher than 1.0. Thus, the model increases the max to 10.0.

Finally, the model sometimes increases the max number of global mappings. By default, SME returns at most three mappings. During shape comparison, three mappings is insufficient, as there may be more than three transformations between the shapes. Thus, the max is increased to five for shape comparison.

3.2 Perceptual Sketchpad

The Perceptual Sketchpad (PSketch) is my extension to CogSketch. Beginning with objects provided by the user, it can, on demand, generate a three-level hierarchical representation. At each level, it identifies the 2-D perceptual features a person might notice. Its qualitative representations are symbolic and structural, meaning they can be compared using SME. At the edge level, it also produces a *quantitative* representation, describing the location, length, orientation, and curvature of each edge.

PSketch models perception in two steps: perceptual organization and perceptual encoding. During perceptual organization (Palmer & Rock, 1994), one identifies the elements at a particular level (e.g., the edges). During perceptual encoding, one encodes the elements' qualitative attributes and relations.

I have suggested representations are constructed bottom-up and accessed to top-down (Hochstein & Ahissar, 2002). However, CogSketch provides ready access to the intermediate object level. Thus, PSketch begins at this level. Given these objects, PSketch can segment an object into edges to produce the edge level, or group objects together to produce the group level. The order of operations is as follows: 1a) edge-level organization, 1b) edge-level encoding, 2) object-level encoding, 3a) group-level organization, 3b) group-level encoding. Thus, aside from beginning with the objects, the model simulates a bottom-up perceptual process.

PSketch's representations depend on the user's initial segmentation of the image. Thus, it is not an entirely autonomous model. The experimenter is responsible for providing consistent, reasonable starting objects. However, the model automatically moves from the object level to the edge and group levels and performs perceptual encoding at each level. Furthermore, sometimes the model can identify new objects by updating the segmentation at the object level. This process of perceptual *reorganization* is covered in section 5.

I now describe the organization and encoding processes in turn.

Perceptual Organization

PSketch performs perceptual organization at each hierarchical level.

Edges

PSketch computes edge-level representations using a single object as input. Each object is represented in CogSketch as a set of *polylines*, lists of points describing strokes drawn by the user. The model segments polylines into perceptually salient edges by identifying junctions where two or more edges meet (Biederman, 1987). Junctions between two edges are identified by discontinuities in the curvature (Lowe, 1989). For example, a square shape would have four strong discontinuities where the four edges meet. See (Lovett et al., 2009b) for details on the segmentation algorithm.

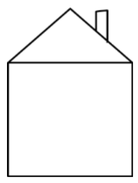


Figure 3.3 Sketch of a house.

After edges have been identified, they are grouped into cycles. A cycle is a series of consecutive connected edges that closes on itself. Thus, any closed shape will produce an edge cycle corresponding to its exterior. Exterior cycles play a key role in perceptual encoding (Hoffman & Richards, 1984). Internal cycles convey less information about an object's 2-D shape, and are not used. For example, consider the house in Figure 3.3. If the user drew this as a single object, its edge-level representation would describe the edges along its contour. While these edges capture its overall shape, they don't describe the three parts of the house: the building, the roof, and the chimney. An intermediate representational level that described the three internal edge cycles would support richer understanding of complex 2-D objects (McLure et al., 2011).

Objects

At the object level, the user provides the elements. However, PSketch can automatically detect objects in three cases. Firstly, objects that intersect, have the same thickness and color, and aren't closed shapes are grouped together. For example, if two intersecting line segments have been added as separate objects, they will be grouped to form a single 'X'-shaped object.

Secondly, PSketch can recognize texture patches, i.e., patterns of repeating elements (Julesz, 1984). Currently, PSketch recognizes basic textures with parallel lines (see Figure 3.4 for examples). If the lines mostly fill another object's outline, the texture is assigned to that object (Figure 3.4B). Objects can have multiple overlapping textures (Figure 3.4C).

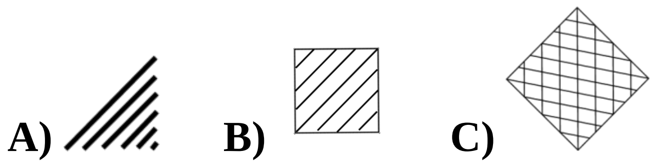


Figure 3.4 Examples of texture patches.

Finally, PSketch can detect space objects. Space objects are defined by negative space. PSketch detects them conservatively: a space object must be convex, it must be surrounded by solid edges, and it must be the only negative space inside a larger containing object. See Figure 3.5 for examples—in each case, the white space is converted to a space object.

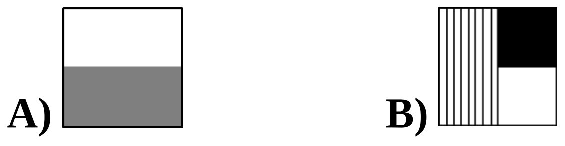


Figure 3.5 Examples of space objects.

Groups

There are several requirements for grouping objects together. These are based upon the Gestalt grouping rules (Palmer and Rock, 1994), which describe how people tend to group objects in their visual field:

1) Similarity: Objects should be the same shape and size.

2) Proximity: Objects should be equally distant from each other.

3) Good continuation: Objects should be axis-aligned (Palmer, 1980). That is, a line connecting the shapes should run along their internal axes.

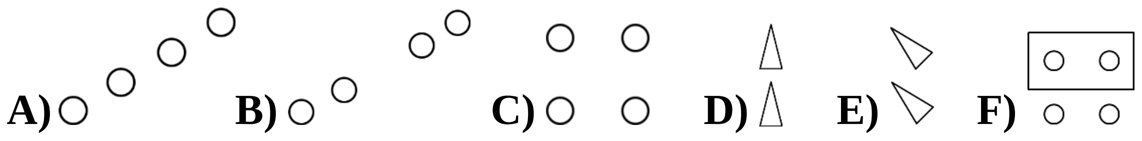


Figure 3.6 Grouping examples.

Similarity. At the object level, perceptual encoding (see below) computes several qualitative features describing each object's shape. Two objects' shape features must be identical before they are grouped together. Additionally, they must be approximately the same size. This is determined by comparing their areas, provided they are closed shapes. For objects that are not closed shapes (e.g., single lines, X's), measuring size is more difficult. Thus, grouping is currently implemented only for closed shapes. Future work will hopefully produce more general grouping criteria.

Proximity. Grouping begins by finding a pair of similar objects that are minimally distance. Then, objects are added incrementally, so long as the new object is a similar distance from some group member. Thus, Figures 3.6A and 3.6C will produce a single group, while Figure 3.6B will produce two groups, segmented by distance.

Good continuation. Two objects can be grouped only if their axes are aligned with a line connecting them. Axes may be either axes of symmetry or axes of elongation. For example, in Figure 3.6D, a line connecting the objects travels through their axes of symmetry *and* their axes of elongation. In Figure

3.6E, it runs through neither, so the objects would not be grouped. Note that circles have an infinite number of symmetry axes, so two circles will always be axis-aligned.

As with proximity, the good continuation check is performed for pairs of objects, not for all the objects together. Thus, the circles in Figure 3.6C can be grouped because each circle is axis-aligned with another circle, even though the four circles do not produce a single line together.

Topological relations between objects constrain grouping further. In Figure 3.6F, the upper circles are grouped separately because they're inside a container. Other topological relations can also disrupt grouping—for example, if an object contains another object, it cannot be grouped.

PSketch uses strict guidelines for grouping, whereas a person might group objects more flexibly. Thus, it produces a conservative representation of the groups in an image. In many cases, objects won't be grouped at all. These individual objects are still included in the group-level representation. If no groups are found in an image, its group-level representation will be identical to the object level.

Perceptual Encoding

At each level in the hierarchy, PSketch generates a qualitative representation describing the elements and the spatial relations between them. The representations are determined by the qualitative vocabulary, the list of terms that can be encoded. Choosing a vocabulary is quite difficult, as there is a near-infinite range of features that might be encoded. Thus, several considerations have gone into choosing our qualitative vocabularies.

First, the vocabularies should only cover features that would be reasonably salient to a person. Thus, psychological research on human vision served as motivation for much of the vocabulary. There is evidence that people are sensitive to many qualitative features, including an object's alignment with the horizontal or vertical axes (Appelle, 1972); parallel and perpendicular lines (Abravanel, 1973; Chen & Levi, 1996; Rosielle & Cooper, 2001); concave corners (Elder & Zucker, 1993; Ferguson et al., 1996); one object's relative location inside another object (i.e., which quadrant of a circle a dot is located in)

(Huttenlocher, Hedges, & Duncan, 1991), and relations describing relative position (above/below, right/left) or topology (on/off) (Kosslyn et al., 1989).

Second, the vocabularies are constrained by the tasks I am modeling. Encoding every feature a person could notice may be unfeasible, so I have taken a conservative approach. PSketch encodes the minimal set of features necessary for human-level performance on my target tasks. The vocabularies could be expanded to cover more tasks in the future.

Third, the vocabularies describe non-accidental features (Biederman, 1987). These are features that are unlikely to have occurred by chance, or due to a particular viewing angle. For example, two randomly chosen objects are unlikely to be overlapping. Thus, PSketch does not encode the absence of an overlapping relationship. However, it *does* encode the presence of such a relationship. Similarly, it does not encode an arbitrary angle between two edges, but it does encode that they are parallel or perpendicular.

Fourth, when possible, PSketch encodes the same features at multiple levels in the hierarchy. For example, an edge might be straight or curved. An object can also be straight or curved. Whenever a feature holds for all the edges in an object, that feature "bubbles up" from the edge level to the object level. This can be an edge attribute, such as straightness, but it can also be a relation between edges: if every edge corner in an object is convex, it is a convex object. Perhaps less intuitively, if every corner is perpendicular, it is a perpendicular object.

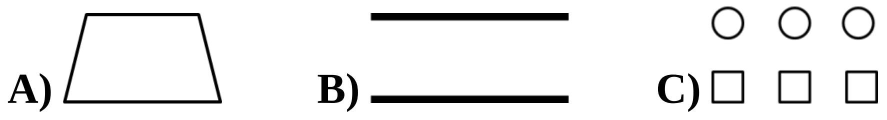


Figure 3.7 Examples of parallel edges, parallel objects, and parallel groups.

Fifth, some relations are shared between levels. For example (Figure 3.7), at the edge level, two edges can be parallel. At the object level, two single-edge objects can be parallel. At the group level, two groups can also be parallel.

We can distinguish between two types of qualitative representations: orientation-invariant, and orientation-specific. In the previous chapter, I suggested that shape representations (the edge level) are often orientation-invariant, allowing for comparison across rotations and other transformations. In contrast, image representations (the object or group level) may be more orientation-specific, representation features such as "right of." However, this likely depends on the task. It should be possible to focus on orientation-invariant or orientation-specific features at any level. PSketch can produce either representation type.

Below, I briefly discuss thresholds and then describe perceptual encoding at each hierarchical level.

Thresholds

Building qualitative representations from quantitative information requires thresholds. For example, a threshold determines at what hue a color changes from "red" to "orange." However, the more thresholds used, the more parameters a model contains, and the weaker it is. Therefore, PSketch has three general thresholds that are used repeatedly, both for constructing representations and for comparison. The first is for comparing orientations, e.g., to see it two edges are parallel. The second is for comparing quantities, e.g., to see if two edges are the same length. The third is for comparing locations, e.g., to see if two edges are located in the same relative location in their respective objects.

These thresholds are malleable: they can be made stricter or looser, depending on the needs of the task. For consistency, I choose a strictness for each threshold and keep it constant across all the tasks I model. Even with a strictness set for the task, certain operations use looser thresholds than others. Orientation comparison is stricter for two edges within a shape than for two edges in separate shapes. This is because one should have greater precision when comparing edges that are touching or proximate than when comparing edges that are farther apart.

While these thresholds are used whenever possible, they are not the only thresholds in the model. Some qualitative relations are computed by CogSketch proper (relative location, topological relations). These relations use their own thresholds which are not available for tweaking. There is also a threshold for determining that three points are collinear, i.e., they form a straight line. Additionally, I use hard-coded thresholds for computing colors from RGB values.

Edges

The input to edge-level encoding is the edges, junctions between edges, and edge cycles identified during perceptual organization. The output is an edge-level description with: 1) attributes that describe a single edge; 2) simple edge relations that describe pairs of edges; and 3) edge cycle relations that describe the edges in a cycle.

Edge cycles are key in representing an object's shape. Therefore, they are encoded as second-order relations, i.e., terms that take other expressions as their arguments. In contrast, simple edge relations are first-order relations that take edges as their arguments. This gives the edge cycle relations more weight in SME's structural comparison.

Attributes	Simple Edge Relations	Edge Cycle Relations	Edge Cycle Arguments
• PerceptualEdge • StraightEdge/ CurvedEdge/ EllipseEdge • length(Tiny/Short/ Medium/ Long) • axisAligned	• edgesPerpendicular • edgesParallel • edgesCollinear • elementsConnected • elementsIntersect • elementIntersects	• cycleAdjacentAngles • adjacentAcuteToObtuseAngles/ adjacentObtuseToAcuteAngles • perpendicularCorner	• convex/ concaveAngleBetweenEdges
		• parallelEdgeRelation • collinearEdgeRelation	• disconnectedEdges

Table 3.1 Orientation-invariant qualitative vocabulary for edges.

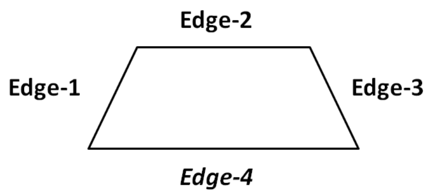
	(**cycleAdjacentAngles** (**convexAngleBetweenEdges** Edge-2 Edge-3) (**convexAngleBetweenEdges** Edge-3 *Edge-4*)) (**cycleAdjacentAngles** (**convexAngleBetweenEdges** Edge-3 *Edge-4*) (**convexAngleBetweenEdges** *Edge-4* Edge-1)) (**cycleAdjacentAngles** (**convexAngleBetweenEdges** *Edge-4* Edge-1) (**convexAngleBetweenEdges** Edge-1 Edge-2)) (**adjacentAcuteObtuseAngles** (**convexAngleBetweenEdges** *Edge-4* Edge-1) (**convexAngleBetweenEdges** Edge-1 Edge-2)) (**adjacentObtuseAcuteAngles** (**convexAngleBetweenEdges** Edge-2 Edge-3) (**convexAngleBetweenEdges** Edge-3 *Edge-4*)) (**parallelEdgeRelation** (**edgesDisconnected** Edge-2 *Edge-4*))
(**PerceptualEdge** *Edge-4*) (**StraightEdge** *Edge-4*) (**axisAligned** *Edge-4*) (**lengthLong** *Edge-4*) (**edgesParallel** Edge-2 *Edge-4*)	

Figure 3.8 Example representation with all expressions about Edge-4 in a parallelogram.

Table 3.1 gives the qualitative terms for orientation-invariant representations. Figure 3.8 shows an example representation for one edge in a parallelogram. I now describe the terms given in Table 3.1, beginning with attributes.

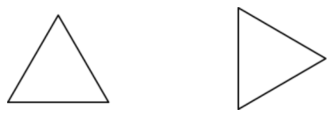


Figure 3.9 A rotation between two triangles.

PerceptualEdge describes all edges. **StraightEdge/CurvedEdge/EllipseEdge** describe a shape's curvedness. Ellipses are curved edges that close on themselves, such as circles. **lengthX** describes qualitative length, relative to the longest edge in the object. **axisAligned** describes a straight edge that is horizontal or vertical. Unlike other terms, **axisAligned** does carry some orientation information.

However, it is useful for aligning objects' bases across rotations. For example, the triangles in Figure 3.9 can be considered 90° rotations because the horizontal base maps to the vertical base.

Simple edge relations are first-order relations between pairs of edges. Some describe relative orientation: **edgesPerpendicular** and **edgeParallel**. **edgesCollinear** describes parallel edges that are also collinear. Other relations describe different ways the edges may intersect without being in an edge cycle. **elementsConnected** describes two edges that meet at their endpoints, while **elementsIntersect** describes two edges that cross each other to form an 'X.' **elementIntersects** describes one edge whose endpoint bisects another, forming a 'T.'

Finally, *edge cycle relations* are second-order terms describing edges in a cycle. Table 3.1 gives these terms along with the first-order relations that serve as their arguments. Most use the first-order relation **convex/concaveAngleBetweenEdges**, which describes the convexity of a vertex between adjacent edges in a cycle. **cycleAdjacentAngles** describes two adjacent vertices, moving clockwise along the cycle. **adjacentAcuteToObtuseAngles/adjacentObtuseToAcuteAngles** describes a change between obtuse and acute angles, again moving clockwise. **perpendicularCorner** describes a single vertex as perpendicular.

The other two edge cycle relations take a different first-order relation as their argument. **disconnectedEdges** describes two edges that are not adjacent in the cycle. The second-order terms **parallelEdgeRelation** and **collinearEdgeRelation** take **disconnectedEdges** as their argument.

Three points should be noted. Firstly, a weakness of this approach: this representation is invariant across rotations (except for **axisAligned**), but it is *not* invariant across reflection. Edge cycle relations always list edges in clockwise order around the cycle; this provides additional structure that aids in SME mappings. However, a mirror reflection would reverse the order of edges in a cycle. Thus, this representation does not support comparing an object to its mirror reflection, a key feature of Biederman's (1987; Hummel & Biederman, 1992) geon structural descriptions. This problem is easily overcome; when the model is looking for reflections, it transforms the qualitative representation by reversing the order of edges in cycles. This requires two separate SME comparisons: one looking for rotation, and one looking for reflection. Admittedly, this may diverge from the way humans perform shape comparisons.

Secondly, relative orientation (e.g., **parallelEdges**) is encoded both as simple edge relations and as edge cycle relations. Simple edge relations are important because two edges can be parallel or perpendicular even if they don't share an edge cycle. Edge cycle relations are important because, as second-order terms, they receive greater weight in SME's mapping.

Thirdly, it is possible for an object to have many parallel and perpendicular edges. To keep the representation manageable, these relations are restricted to only hold between adjacent edges. For perpendicular, this means the edges must touch. For parallel, it means that when the edges are in a cycle, they must be across from each other.

Some tasks may require a richer, orientation-specific representation. PSketch can supplement the terms above with additional terms to describe orientation, relative position, etc. See Table 3.2 for these terms.

Attributes	Simple Edge Relations	Edge Cycle Relations
• VerticalEdge • HorizontalEdge • ObliqueEdge-Upward/Downward • CurvedEdge-Right/Left/Up/Down Bumped	• rightOf/above • edgesCurveCompati ble • edgeCurveCompatibleWith	• leftToRightCorner/rightToLeftCorner/ topToBottomCorner/bottomToTopCorner • verticallyOrientedCorner/ horizontallyOrientedCorner

Table 3.2 Additional orientation-specific vocabulary for edges.

The attributes describe the orientations of straight and curved edges. For example, **ObliqueEdge-Upward** describes a diagonally-orientation edge that slopes upward moving from left to right, and **CurvedEdge-RightBumped** describes a curve whose center points in the rightward direction.

The simple edge relations describe the relative orientation of two edges, or the compatibility of two curves. **edgesCurveCompatible** indicates that two edges' curvature is in the same direction as the angle that connects them. **edgeCurveCompatibleWith** indicates only one of the edges' curvature matches the angle. Curve compatibility is not strictly orientation-specific; however, it is an added detail not included in the sparser orientation-invariant representation.

The edge cycle relations again take **convex/concaveAngleBetweenEdges** as their argument. They describe the relative position of two edges meeting at a corner, or a corner's orientation. For example, if two edges form a corner that points rightward, the corner is a **horizontalOrientedCorner**.

Basic Attributes	**Edge-Based Attributes**	**Symmetry Attributes**	**Location Attributes**
• 2D-Shape-Generic	• 2D-Shape-Convex	• Symmetric-Shape	• Centered-Element
• 2D-Shape-Open/Closed	• 2D-Shape-Curved/Straight/Ellipse	• Perpendicular-Symmetric-Shape	• OnTop-Element ^{O}
• 2D-Shape-Forked	• 2D-Shape-Axis-Aligned^{O}	• Multiply-Symmetric-Shape	• OnLeft-Element^{O}
• 2D-Shape-Oblique ^{O}	• 2D-Shape-Perpendicular	• Fully-Symmetric-Shape	
• VerticalEdge/ HorizontalEdge^{O}		• Non-Elongated-Shape	
• Dot-Shape			
• Implicit-Shape			

Size Attributes	**Color/Texture Attributes**
• narrowEdges/wideEdged	• (ObjectsColoredFn *color*)
• (Tiny/Small/Medium/Large)SizeShape	• (ObjectsBorderColoredFn *color*)
	• TexturedObject

Spatial Relations	**Alignment Relations**	**Transformation Relations**
• rightOf/above ^{O}	• parallelElements	• reflectedShapes-XAxis
• onRightHalfOf/onLeftHalfOf ^{O}	• perpendicularElements	• reflectedShapes-YAxis
• onTopHalfOf/OnBottonHalfOf ^{O}	• collinearElements	• reflectedShapes
• centeredOn	• centeredOn	• rotatedShapes-90
• elementsIntersect		• rotatedShapes-180
• elementsOverlap		• rotatedShapes
• elementContains		

Table 3.3 Object-level qualitative vocabulary. Terms marked with an '^{O}' are orientation-specific.

Objects

Table 3.3 gives the qualitative vocabulary for objects. Object-level representations are normally orientation-specific, unlike edges. Thus, the table gives the full set of terms. Orientation-specific terms, marked with an ‘^{O},’ are excluded when an orientation-invariant representation is requested.

I begin with basic attributes. **2D-Shape-Generic** describes all objects except space objects, which are instead **Implicit-Shape.** Objects can be open or closed shapes. **2D-Shape-Forked** describes an object containing a forked junction, where three or more edges meet, such as a ‘T’ or ‘X’ shape. Objects consisting of a single straight edge can be vertical, horizontal, or oblique. Finally, **Dot-Shape** describes an object that is too small for shape information to be computed.

Edge-based attributes describe the edges within an object. As mentioned above, any time a feature holds for all the edges in an object, or for all the corners between edges, that feature “bubbles up” to the object level.

Symmetry attributes describe the axes of symmetry in a shape. Symmetry is computed based on the edges within an object (see below)—thus, it is a necessarily bottom-up feature. The different attributes refer to the number of symmetry axes, as well as the relations between axes. For example, an equilateral triangle has multiple symmetry axes, so it is a **Multiply-Symmetric-Shape**. A rectangle has two perpendicular axes, so it is a **Perpendicular-Symmetric-Shape**. A square or circle has many axes, some of them perpendicular, so it is a **Fully-Symmetric-Shape**.

Note that the last symmetry type, **Non-Elongated-Shape**, is not a form of axial symmetry; it simply indicates the absence of any axis of elongation. It is included here because many symmetric shapes (squares, circles, equilateral triangles), are non-elongated. When symmetry information is unavailable, this feature is a cue that a shape is likely to be symmetric. When symmetry information *is* available, it takes precedence over this attribute.

The next attributes describe relative location and relative size. Location is relative to the other objects in an image. In contrast, size is typically computed across all images being considered. Thus, objects in different images with the same size will match. Relative size is encoded both for the width of an object's edges and (for closed shapes) for its overall area.

The remaining attributes describe appearance, rather than shape. An object may have separate colors for its border and its fill. Additionally, it may be classified as a **TexturedObject** if it is filled with a texture

Spatial relations (Table 3.3, bottom left) are the basic relations CogSketch computes between objects. **rightOf** and **above** describe one object's location relative to another. The next three relations describe relative location when one object is inside another. For example, **onRightHalfOf** describes a contained object located on the right side of the container object, whereas **centeredOn** described a contained object centered on the container object. The final three relations are topological. **elementsIntersect** means the objects' borders intersect, **elementsOverlap** means their areas partially overlap, and **elementContains** mean one object is inside the other. These relations are not mutually exclusive (Lovett & Forbus, 2009b); it is possible for two of them to hold.

As with edges, some spatial relations require that the two objects be adjacent. **rightOf/above** only hold if no third object lies between the two, and **(elementContains** X Y**)** only holds if no third object contains Y and is contained by X.

Alignment relations describe objects that consist of a single edge. Like edges, these objects can be parallel or perpendicular. Additionally, if one object is an edge, another object can be centered on it, or an object can be collinear with it if it lies along an extension of that edge. Note that **centeredOn** is thus both a spatial relation and an alignment relation.

The remaining relations describe transformations between two objects' shapes. One object might be a reflection or a rotation of another. As with symmetry, shape transformations are computed between objects' edge-level representations.

Note that there are sometimes multiple transformations between two shapes. For example, consider two squares. The shapes are identical, but there are also rotations or reflections of one square that would produce the other. In such cases, the model encodes only the simplest shape transformation. Here, the simplest transformation is identity, so no rotations or reflections would be encoded. I discuss shape transformation in greater detail below.

Groups

Recall that the group level may contain individual objects, as well as groups. The objects are represented as above. However, groups are represented using only a subset of the object-level terms. Firstly, groups get only a single attribute, **ProximalShape-Group**, which applies to all groups. Secondly, three spatial terms may describe groups: **above**, **rightOf**, and **elementContains.** For example, a group may be right of an object or contained within an object (Figure 3.6F). Finally, some groups are rows of objects (Figure 3.6A). Such groups may be treated as individual edges for alignment relations. Thus, two rows of objects may be parallel, or an object may be collinear with a row if it lies along that row's extension (Figure 3.6B).

3.3 Image Comparison

I model image comparison using SME over qualitative spatial representations. This process generates a similarity score, a set of commonalities, and a set of differences. Overall, there are two types of image comparison: basic and analytic. In basic image comparison, a single SME comparison is computed between the images. In analytic image comparison, the model performs analytic top-down processing: after comparing the images, it compares each corresponding pair of objects to look for shape transformations between them. I describe each of these in turn, using Figure 3.10 as a running example. The objects in Figure 3.10 are labeled for clarity. Upper labels refer to large objects, and lower labels refer to small objects.

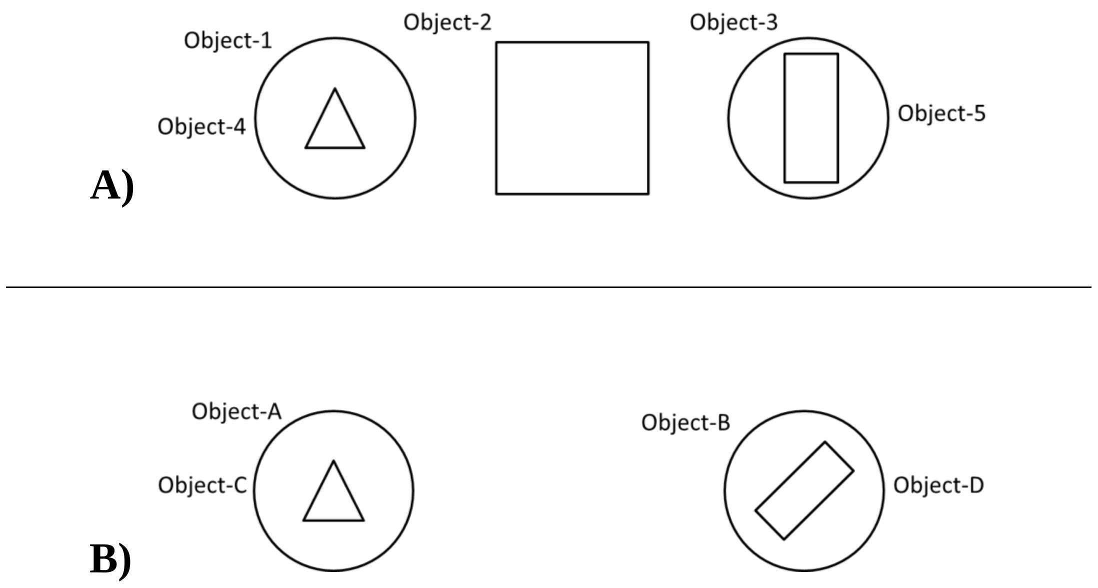


Figure 3.10 Two images that might be compared, with the objects labeled.

Basic Image Comparison

Basic image comparison requires two steps: 1) compare the image representations using SME; 2) evaluate any differences detected. Differences are evaluated because in some cases, features may not be encoded. For example, in Figure 3.10A, Object-3 is right of Object-1. However, this relation is not encoded because the objects aren't adjacent. A comparison with 10B may then suggest a difference: in 10B one circle is right of the other, but in 10A it isn't. By returning to the quantitative information (the sketch) and evaluating this relation, the model determines that (**rightOf** Object-3 Object-1) is true, and therefore this isn't a difference between the images.

A -> B Inferences	B -> A Inferences
(**MediumSizeShape** Object-D) (**2D-Shape-AxisAligned** Object-D) (**rightOf** Object-B (:skolem Object-2)) (**rightOf** (:skolem Object-2) Object-A))	(**SmallSizeShape** Object-5) (**rightOf** Object-3 Object-1)

Table 3.4. Candidate inferences between images 10A and 10B.

Let us consider this example in detail. Suppose images 10A and 10B are encoded at the object level. The SME comparison between them produces a mapping in which the circles correspond (Object-1⇔ Object-A, Object-3 ⇔ Object-B) and the triangles and rectangles correspond (Object-4 ⇔ Object-C, Object-5 ⇔ Object-D). This mapping also includes several candidate inferences (Table 3.4). Recall that candidate inferences are expressions in one representation that fail to map to the other. For example, Object-5 is a **MediumSizeShape**, but the corresponding Object-D is a **SmallSizeShape**. Therefore, one of the inferences going from A to B is that Object-D should have been **MediumSizeShape**.

Overall, the candidate inferences express differences in objects' shape attributes and in their spatial relations. The rectangle changes its shape attributes; it goes from being medium-sized and axis-aligned to being small-sized. The circles change their spatial relations. In 10A, Object-3 is right of the square, and Object-1 is left of the square. In 10B, neither of these relations holds—the :skolem indicates that the square object doesn't even exist in 10B. However, in 10B, one circle is right of the other. In 10A, this relation isn't encoded because the circles aren't adjacent, thus giving rise to the last candidate inference.

During evaluation, the model checks each of these expressions to see if they may actually be true. (**rightOf** Object-B (:skolem Object-2)) can't be true, as the :skolem indicates the second object doesn't exist. However, (**rightOf** Object-3 Object-1) is true—one circle is right of the other in 10A. Therefore, this expression is added to 10A's representation, and the mapping is updated, removing this difference.

The model performs a similar check for transformation relations. For example, suppose image A contains two circles, while image B contains rectangles at 90° rotations from each other. One of the B->A inferences will be (**rotatedShapes-90** objA objB). In this case, the model will check the circles in image A and determine that, yes, there could be a rotation between them. The rotation wasn't encoded because the model prefers perfect identity matches to rotations (see below).

Resolving Ambiguity

Sometimes SME finds multiple equally good mappings. The model employs two strategies for resolving ambiguity to pick the preferred mapping. Firstly, consider Figures 3.11A and 3.11B. Based purely on object-level shape features, the object in 11A maps equally well to either object in 11B. In this case, the model performs shape comparisons between the 11A object and the 11B objects (see below). It chooses the mapping that produces the best shape match.

Secondly, consider Figures 3.11C and 3.11D. Here, all shapes match equally well, and the mapping truly is ambiguous. The model imposes a left->right and top->bottom ordering in such cases. Here, the circle in 11C will map to the leftmost circle in 11D. I do not claim this is a better mapping than the alternative. It simply ensures the model is consistent.

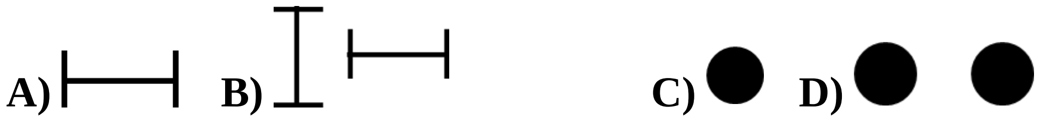


Figure 3.11 Two ambiguous mappings.

Analytic Image Comparison

The basic shape comparison provides limited information. It indicates that the rectangle in Figure 3.10A has changed, becoming smaller and losing its axis-alignment, but it doesn't determine exactly *how* the rectangle has changed. In this case, the rectangle has rotated, but it might instead have been replaced with a different shape, such as a triangle. To determine how shapes change between images, one must compute transformations between their objects. This is analytic image comparison.

Analytic image comparison implements analytic top-down comparison, as described in the previous chapter. It is a recursive process, in which correspondences at one level guide comparisons at the next level down. Its purpose is to replace concrete shape features with a more abstract, more concise representation of how objects change between images.

The process begins with basic image comparison. The initial comparison produces a set of corresponding elements (objects or groups) in the two images. The model then iterates over each corresponding pair and compares their shapes (see the following section for details). The comparison either produces a transformation between the elements or determines that they are entirely different. In our running example, the rectangle rotates and shrinks between Figures 3.10A and 3.10B. The other corresponding objects are identical.

Afterwards, the model removes all shape and position attributes from the representations, producing a more abstract image representation. The model updates the mapping score such that corresponding objects are scored only for whether they have identical shapes. For the differences, it represents any shape transformations between objects. In our running example, the attribute inferences would be replaced with two differences: the rectangle becomes smaller, and it rotates. Differences in colors and spatial relations, such as **rightOf**, are retained.

The model also compares textures in corresponding objects. Textures are considered the same if either they have identical orientations (e.g., Figure 3.12A), or the same transformation would align both the shapes and the textures (e.g., reflecting over the x-axis in Figure 3.12B).

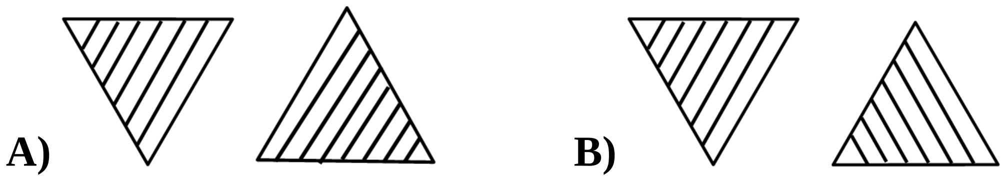


Figure 3.12 Object pairs whose textures would be considered identical.

This approach presents two related challenges: 1) What difference do we encode when one object changes to an entirely different object (e.g., a square to a circle)? 2) What difference do we encode when one texture changes to a different texture? There are likely several solutions to these problems. However, to support spatial problem-solving, it is useful to encode more information than simply "the object changed shape." If a square changed to a circle, it may be useful to actually encode "a square changed to a circle." The model, however, has no pre-existing knowledge of the shape types it will encounter. Instead, it must learn the shape types over the course of a problem. Similarly, it must learn texture types, where a texture is identified by its orientation. The model handles this with shape and texture registries. I explain the shape registry below. Texture registries are similar but simpler.

Shape Registry

The shape registry stores classes of shapes. When an object is registered, it is assigned a *shape class* and a *strict shape class*. Objects in the same shape class have a valid shape transformation between them, a rotation/reflection/scale change. Objects in the same strict shape class are identical. Every shape class and strict shape class has a unique id. Thus, an object can be assigned an attribute such as (**ShapeTypeFn** *class-id*) or (**StrictShapeTypeFn** *class-id strict-id*). Each shape class also has a prototype, the first object encountered of that class. Strict shape classes are stored with their transformation from their class' prototype.

When a new object is registered, it is compared to an exemplar from each shape class (typically the prototype, although there is a preference for exemplars close in size). If it matches a shape class, it is compared to an exemplar from each strict shape class within that shape class. If a match is found, the object is added to that shape class and strict shape class. Otherwise, a new shape class or strict shape class is created, with the object as the prototype.

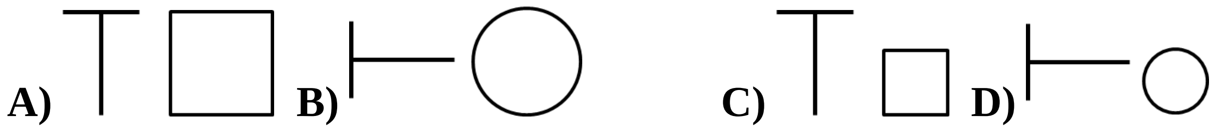


Figure 3.13 Shape change examples.

Returning to image comparison, suppose the model compares Figure 3.13A to 3.13B. The corresponding 'T' shapes are rotations, but there is no transformation between the square and the circle. Therefore, each of the shapes is registered in the shape registry and assigned a shape class. Because these are the first shapes encountered, they are the prototypes for their shape classes. Thus, the model would represent the difference as a change from (**ShapeTypeFn** 0) to (**ShapeTypeFn** 1). This is equivalent to remembering, "This one shape (square) changes to this other shape (circle)."

Now, the model compares 13C to 13D. Again, there is a change from a square to a circle. These shapes are registered, and in each, the shape is a smaller version of the prototype in its shape class.

Because the two objects have analogous transformations from their prototypes, the difference can again be represented as a change from (**ShapeTypeFn** 0) to (**ShapeTypeFn** 1). If they were not analogous transformations (e.g., a big square to a small circle), the model would use their strict shape classes to represent the difference.

3.4 Shape Comparison

During visual encoding and comparison, it is useful to see how two objects' shapes relate to each other. Shape comparison determines whether one of the following is true: 1) the objects' shapes are identical; 2) there is a transformation between the shapes, such as a rotation; 3) there is a systematic difference between the shapes, i.e., a deformation such as stretching the edges in one direction. It is possible for more than one of these to hold between shapes. For example, consider two identical squares. Several rotations or reflections of one square would produce the other. However, the model attempts to identify the simplest transformation between the shapes. Thus, identity is preferred to rotation. Similarly, a small rotation (such as 45°) is preferred to a large rotation (such as 135°).

Shape comparison is used in two places. Firstly, during object-level encoding, the model encodes rotations and reflections between objects' shapes *within* an image. Secondly, during image comparison, the model identifies differences between shapes *across* images. Within an image, the model checks every pair of objects, but across images, it only considers the corresponding objects. Therefore, shape differences across images are represented more richly, and the model attempts a greater range of shape comparisons.

Shape comparison is based on the previous chapter's theory of mental rotation. The model compares objects' shapes by aligning their edge-level representations. Recall that there are two representations at the edge level: a qualitative representation, and a quantitative representation which describes each edge's location, orientation, length, and curvature. Comparison proceeds in three steps: 1) Compare qualitative representations with SME. Get corresponding edges. 2) Select a pair of corresponding edges. Compute the quantitative transformation between them (e.g., a rotation + a change in scale). 3) Apply the transformation to all the edges in the first object's quantitative representation. Check whether the first object's edges are now the same as the second object's edges.

Below, I describe these steps in detail for rotations. I then explain how the model handles others transformations. Finally, I discuss how the transformations are ordered from simplest to most complex.

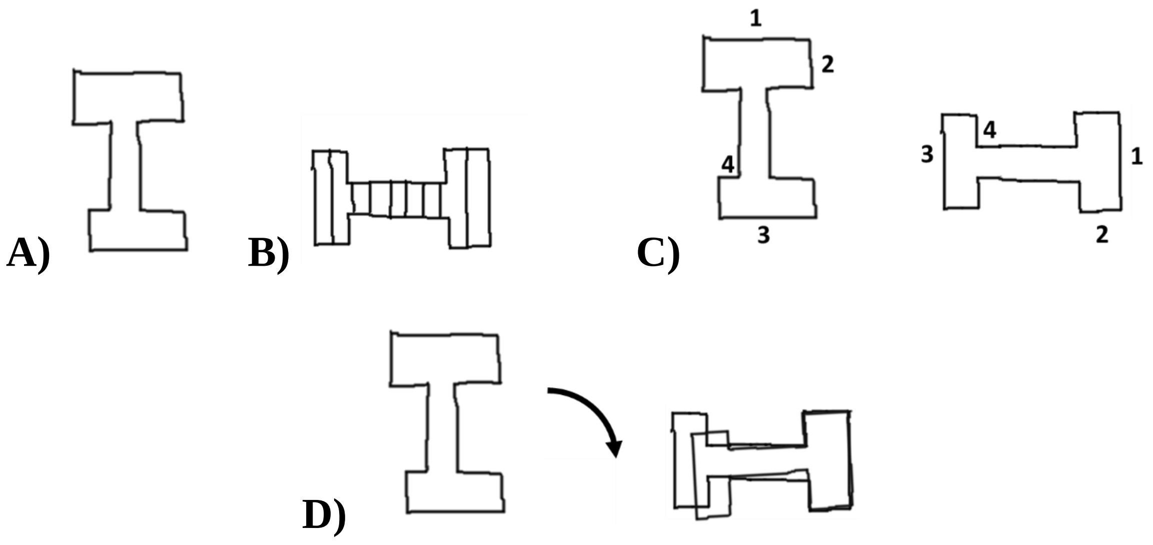


Figure 3.14 Shape rotation example.

Rotation

Figure 3.14 shows two "I" shapes at different orientations. To find a rotation, the model does the following:

1) Compare qualitative representations with SME

The model uses the orientation-invariant edge-level representations to find rotations. Recall that a) these representations are rotation-invariant but *not* reflection-invariant because edge cycle relations encode a clockwise ordering of edges; and b) these representations encode only each shape's outer contour. The interior edges in Figure 3.14B will be ignored.

The model makes one change to SME's default parameters: instead of looking for up to three mappings between the representations, SME looks for up to five. This is done because some regular shapes, e.g., squares, may have more than three valid transformations between them.

SME produces 1-5 mappings between the edges. The model picks out the top-scoring mapping, along with any other mappings with an equal score.[3] Any mappings that fail to match between all edges are removed.

For Figure 3.14, SME finds two equal mappings between the edges. These mappings represent 90° rotations in each direction. Figure 3.14C shows corresponding edges for one mapping—note that all edges correspond, but only a few are labeled for simplicity.

Steps 2) and 3) are performed for each SME mapping, as each mapping is a different potential transformation.

2) Select a corresponding pair and compute a quantitative transformation

The model now moves from the qualitative to the quantitative. This is easy because each qualitative edge label indicates a quantitative edge (with location, orientation, length, etc).

The model picks an edge pair to compute a transformation. For rotation, the model always picks the edge pair with the greatest combined length. This is done because longer edges might be more salient to a person, and they provide the most information about the shape transformation. In Figure 3.14C, the model chooses edge pair 1.

Going from 14A to 14B, the model computes a rotation about edge 1's center, accompanied by a scale change. Here, the edge rotates about 90° clockwise and becomes slightly shorter.

[3] Scores whose difference is less than .001 are considered equal. I use this small similarity threshold because it is possible in SME for equivalent mappings to receive slightly different scores.

3) Apply the transformation to all edges and evaluate

The model applies the transformation to all the edges in 14A. It then iterates over each edge and checks whether it matches the corresponding edge in 14B. This is based on several criteria: relative location in the image, orientation, length, and curvature. The first three criteria use thresholds. As mentioned above, the model uses looser thresholds for these comparisons between objects than within an object. For curvature, the model simply requires that if the edges are curved, they curve in the same direction.

Figure 3.14D shows the result of rotating and scaling 14A. Edge 1, the edge that guided the rotation, aligns perfectly. However, as edges get farther from edge 1, the match deteriorates. The model handles this by degrading the thresholds for relative location and length. Thresholds degrade linearly with distance from a standard (edge 1's location or length), such that when they are maximally distant, the threshold will be 50% looser. For relative location, this means that edge 3, the farthest edge from edge 1, has the loosest location threshold. For length, this means that edge 4, the shortest edge, has the loosest length threshold. The latter is particularly important—as Figure 3.14C shows, while edge 1 gets slightly shorter, edge 4 gets significantly longer.

The output from stage 3) is a list of transformations. Each transformation includes a mapping between edges, a rotation value, and a scale value. Transformations are ordered by the size of the rotation. Rotations of about 0°, indicating identical shapes, will come first, while rotations of about 180° will come last.

Note that this diverges from mental rotation in people because people appear to find the shortest possible rotation first. In contrast, the model finds all rotations and then orders them from shortest to longest. The model might better match human performance if it performed step 2) for all SME mappings, picked out the shortest rotation, and then performed step 3) for just that rotation.

Special cases

Two object types are handled differently: ellipses and dots. Recall that ellipses are closed shapes made of a single curved edge, while dots are objects that are too small for shape details. The model makes the simplifying assumption that these are objects are perfectly symmetrical, and so any rotation of one object will produce the other. When asked for rotations, it returns the four most regular rotations: 0°, 90°, 270°, and 180°.

Other Shape Comparisons

Table 3.5 lists the full set of shape comparisons. As the table shows, different edge-level representations are used when checking for each type. The table also shows the shape relations computed within an image, and the shape differences computed across images. The across-image differences are far richer. Some comparisons, such as deformation, are not even performed with an image, as it would be inefficient to check for a deformation between every pair of objects in the image.

I now discuss the other shape comparisons, focusing on how each is different from rotation.

Reflection

Reflection refers to axial or mirror reflections. For example, returning to Figure 3.14, there is also a reflection over a diagonal axis between the two "I" shapes. Importantly, there are two types of reflections: *spatial reflections*, and *shape reflections*. In a spatial reflection (Figure 3.15A), the axis of reflection lies directly between the two objects. However, in a shape reflection (Figure 3.15B), the locations of the objects are irrelevant. The model requires spatial reflections between objects in the same image, but only shape reflections between objects in differences images.

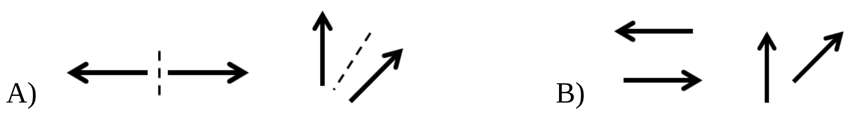


Figure 3.15 A) Spatial reflections. B) Shape reflections.

Comparison Type	**Representation Used**	**Within-Image Relations**	**Between-Image Differences**
Rotation	Orientation-invariant edges	•**rotatedShapes-90** •**rotatedShapes-180** •**rotatedShapes**	•**RotatedShape-45** •**RotatedShape-90** •**RotatedShape-135** •**RotatedShape-180** •**RotatedShape-225** •**RotatedShape-270** •**RotatedShape-315**
Reflection	Orientation-invariant edges, reversed	•**reflectedShapes-XAxis** •**reflectedShapes-YAxis** •**reflectedShapes**	•**ReflectedShape-XAxis** •**ReflectedShape-YAxis** •**ReflectedShape-Diag-Axis** •**ReflectedShape-Other**
Deformation	Orientation-specific edges	--	•**DeformedShape-Longer** •**DeformedShape-Shorter** •**DeformedShape-LongerPart** •**DeformedShape-ShorterPart** •**DeformedShape-AddedPart** •**DeformedShape-RemovedPart**
Symmetry	Symmetric orientation-invariant edges	--	--
Groups	Orientation-specific objects	--	•**ScaledGroup-Larger** •**ScaledGroup-Smaller** •**RearrangedGroup** •**2D-Shape-Group** •**2D-Shape-Generic**
Special Comparison Type	**Inferred from Type**	**Within-Image Relations**	**Between-Image Differences**
Identity	Rotation, Deformation, Group Comparison	--	•**ConstantShape**
Scaling	Rotation, Reflection	--	•**ScaledShape-Larger** •**ScaledShape-Smaller**

Table 3.5 Types of shape comparisons.

Like rotation, reflection begins by comparing the objects' orientation-invariant, edge-level representations. However, as noted above, edge cycle relations order the edges in a clockwise direction.

Mirror reflections cause the order of elements to reverse. Therefore, the model reverses edges in one of the two representations before comparing.

Steps 2) and 3) are essentially the same as rotation. Instead of computing a rotation and a scale change, the model computes an axis of reflection and a scale change. For spatial reflections, the axis must lie between the objects.

Deformation

Whereas transformations preserve an object's shape, deformations change it in some systematic way. The model checks for several deformation types. For simplicity, the model does not check for combinations of deformations and transformations. Instead, it requires that the shapes have identical orientations and scales, with no differences aside from the deformation. Therefore, it can use the richer orientation-specific representations when comparing at the edge level.

For rotations and reflections, the model can compute a single transformation and then apply it to all the edges in the first object. Deformations are more complicated—they do not preserve shape, so there is no single transformation for all the edges. Instead, the model finds the corresponding edges of interest, uses them to determine the deformation type, and then computes a separate transformation for each edge in the first object. This process is more straightforward with closed shapes consisting of a single edge cycle. Therefore, deformations are only computed for such shapes. I now briefly discuss each deformation type.

DeformedShape-Longer/Shorter. A shape becomes longer if the two edges along its central axis lengthen, while other edges remain the same. See Figure 3.16 for examples. Note that other edges can change (e.g., the top edge in 16B becomes shorter), but only as a result of changes to the two critical edges.

For this deformation, the model looks for two corresponding edges pairs that are each changing length by the same ratio. The edges should be on either side of the object's center and should be symmetric across it. Given such a pair, the model computes a few candidate deformations: the two edges may be growing longer in one direction while the other endpoint stays constant (16B), or they may be growing longer in both directions (16C). For some shapes (16A), these are equivalent. For each candidate deformation, the model updates the location, length, and orientation of the other edges, computing a separate transformation for each edge. It applies the transformations to the edges in the first object, and checks whether they now align with the second object.

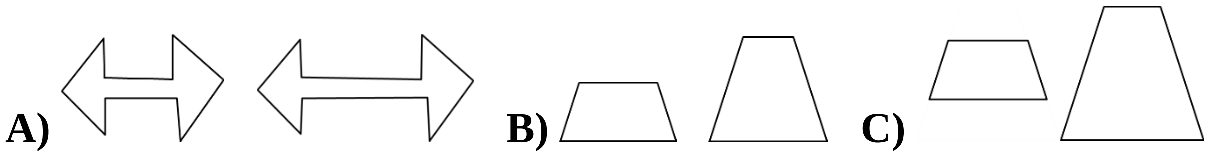


Figure 3.16 Examples of lengthening.

DeformedShape-LongerPart/ShorterPart. A part becomes longer when the first and last edges of the part increase by a constant amount (Figure 3.17). A part can be detected because there will always be a concavity at one end of the part (or sometimes at both ends). The two critical edges should be roughly parallel. Once the edges have been detected, the model computes transformations for the other edges and evaluates.

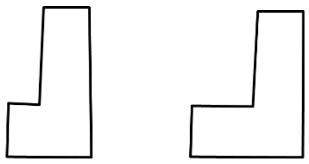


Figure 3.17 Example of part lengthening.

DeformedShape-AddedPart/RemovedPart. There are three ways a part can be added to a shape, distinguished by the convexity of angles and the changes to edge lengths (Figure 3.18). A part can split an existing edge (A), it can extend an existing edge (B), or it can lie between two existing parts, causing the edges on either side to be shortened slightly (C).

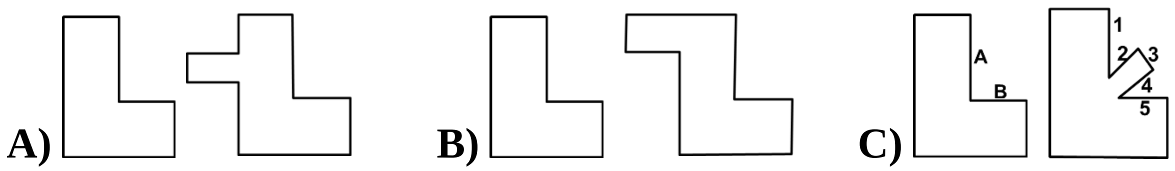


Figure 3.18 Examples of part addition.

Part additions are detected when two cycles have different numbers of edges. The model again identifies the critical edges, the one or two edges on either side of the newly added part. It computes transformations for the edges wherein the critical edges become longer or shorter (or split), and all other edges remain the same.

Note that comparing edge cycles with different numbers of edges can throw off SME's mappings. For example, in Figure 3.18C, SME might map edge A to edge 1 and edge B to edge 2, since the two edges are joined by a concave corner in both objects. However, edge B should actually map to edge 5. This mistake can only happen around the critical edges, the edges adjacent to where the new part is being added. Therefore, the model compares the critical edge correspondences and adjusts them to produce the best quantitative match; here, edge B maps better to edge 5 because they have the same orientation.

DeformedShape-FullShape/SubShape. Finally, one object may be a subshape of the other, meaning it possesses a subset of the other object's edges (Figure 3.19). Unlike the other deformations, this does not require that both objects be closed shapes, as removing some edges will generally open the shape. This deformation is not included in Table 3.5, as it is not generally encoded as a difference between images. However, it is useful in perceptual reorganization (described below).

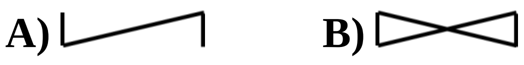


Figure 3.19 Example of a subshape deformation.

Detecting a subshape is simple: one object should have fewer edges, and there should be a perfect identity transformation between those edges and the corresponding edges in the other object.

Symmetry

Symmetry is a special case, wherein an object is compared to itself to identify axes of symmetry, i.e., axes along which the object could be reflected to produce an identical object. The model uses MAGI (Ferguson, 1994), an extension to SME that computes symmetric self-mappings for a single representation. The model also uses a specialized representation scheme, based on symmetric relations that do not impose a clockwise ordering on the edges in a cycle. This representation scheme is also less strict in filtering out non-adjacent parallel edges, as such edges can be a useful guide in identifying symmetry.

Using MAGI and the symmetric edge relations, the model can compute self-mappings such as Figure 3.20 that indicate axes of symmetry. Note that in a symmetry mapping, some edges can map to themselves.

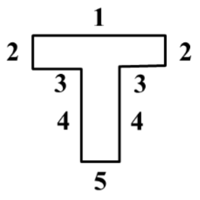


Figure 3.20 Corresponding edges for a symmetry mapping.

Step 2) for symmetry detection is slightly different: when possible, the model finds an edge that maps to itself (e.g., edge 1 in Figure 3.20) to compute the transformation. As with reflection, the model uses the corresponding edge pair (or self-mapped edge) to compute an axis of reflection. Symmetry is always a spatial reflection.

Group Comparison

Unlike the above comparisons, group comparisons relate the shapes of *groups* to each other. The model only considers groups related if they're made up of the same object. Thus, given a pair of corresponding groups, the model will compare an object in one to an object in the other to check whether they are identical. Assuming they are, there are two possible group differences.

ScaledGroup-Larger/Smaller. This is analogous to DeformedShape-SubShape between objects. One group contains all the objects in the other group, in the same configuration, plus some additional objects (Figure 3.21). To detect this, the model compares the groups at the object level to find their corresponding objects.

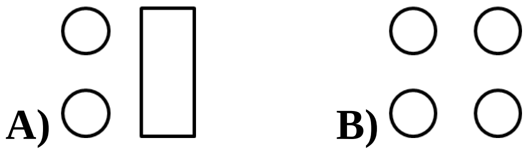


Figure 3.21 Example of a scaled group.

Unfortunately, just comparing the objects in the groups can be insufficient. For example, the two circles in 20A probably correspond to the leftmost circles in 20B, but we only know that because we see that the circles are to the left of the rectangle. Thus, the model compares the groups via the following three steps: 1) break each group down into its objects and compute an updated object-level representation of the image; 2) mark each former group object as important, using a special attribute; 3) constrain SME to only allow these “important” elements to map to other “important” elements. Thus, although the entire images are being compared, the circles in one image are guaranteed to only map to the circles in the other image. Essentially, this is a comparison between just the former group objects, with their relations to other objects in the image constraining the comparison.

Given an SME mapping, the model evaluates it in the way subshapes are evaluated. However, because these are groups, their objects are expected to all be the same size and orientation. Therefore, the model only checks that corresponding objects are in the same relative locations.

RearrangedGroup. If two groups contain the same objects but *not* in the same locations, i.e., there isn’t a scaled group, then they are labeled as rearranged groups.

2D-Shape-Generic/2D-Shape-Group. This is a change from a single object to a group of objects, provided the objects in the group are identical to the single object.

Identity

Identity is the first of two special comparison types. Rather than requiring its own comparison, identity can be inferred from three of the above comparisons. Firstly, during rotation comparison, if the angle of rotation is about 0° and there's no change in scale, the objects are identical. Secondly, during deformation comparison, if the objects contain the same number of edges in corresponding locations, the objects are identical. Thus, either orientation-invariant or orientation-specific representations can support finding identity. Finally, during group comparison, if the groups contain the same number of objects in corresponding locations, the groups are identical.

Scaling

Scaling is the second special type. During rotation and reflection checks, the model includes a scaling factor. If there is a valid transformation, and the scaling factor indicates a change in size, the model encodes the change in scale. Thus, objects might have two differences between them: a rotation and a change in scale.

Order of Transformations

As stated above, the model returns only the simplest transformation between two shapes. This requires a simplicity ordering on the transformations. Based on intuitions, I have used the following ordering: Identity, Scaling, Canonical Reflection (over the x- and y-axes), Rotation, Non-Canonical Reflection. Thus, the most obvious reflections are considered simpler than rotations, while other reflections are considered more complex. Among rotations, transformations are ordered from the shortest rotation to the longest.

This is only a best guess at how transformations should be ordered. However, one of our simulations (geometric analogy) lets us evaluate this ordering. As we shall see, when faced with ambiguous shape comparisons, people do indeed choose canonical reflections over rotations.

3.5 Perceptual Reorganization

The model I have described takes a set of objects identified by a user. It breaks them down into edges and joins them up into groups. It generates hierarchical representations to support both image comparison and object comparison. By moving between levels in the hierarchy, the model can better understand how two images are the same or different.

Because the user provides information at the object level, perception begins at this the intermediate level. Aside from this first step, representations are constructed bottom-up. Perceptual encoding of edges facilitates perceptual encoding of objects, and encoding of objects facilitates organization and encoding of groups.

However, perception is not always bottom-up. For example, Medin, Goldstone, & Gentner (1993) had participants compare ambiguous visual stimuli. They found that people changed their perception of the stimuli to make them more similar. Similarly, the model is able to perform perceptual reorganization, in which it segments an image into a different set of objects to facilitate a comparison.

I have implemented two forms of perceptual reorganization: basic and complex. Basic reorganization requires only a single analytic image comparison, whereas complex reorganization requires multiple comparisons. Thus, the model predicts that complex reorganization will be more effortful for people. I will evaluate this prediction in a later chapter.

Basic Reorganization

Basic perceptual reorganization can be triggered whenever an image comparison produces a subshape deformation (Figure 3.19) or a scaled group transformation (Figure 3.21). These comparison types indicate that two images have identical elements (objects or groups), but in one of the images, that element has extra parts (extra edges or objects). For example, Figure 3.19A contains an object with three edges. Figure 3.19B contains an object with the same edges, plus an additional diagonal edge. Similarly, Figure 3.21A contains a group with two circles, while Figure 3.21B contains a group with those same two circles, plus two additional circles to their right.

Subshapes and scaled groups represent complicated differences between images. It's always simpler when images contain identical objects. Therefore, the model splits the object in 19B into two objects: one that matches 19A, and one with the remaining edge. Similarly, it splits the group in 21B into two groups.

Once reorganization is complete, the model repeats the image comparison. The differences between the images are now more meaningful: in Figure 3.19, an edge is added, while in Figure 3.21, the rectangle is replaced with two circles.

Note that there are several constrains on when an element can be split into two elements. An object can only be split up if its interior is empty; a solid black square or a rectangle filled with a texture could not be split into edges. Also, an element can only be split up if both the resulting elements are coherent: that is, an object's edges are connected, or a group's objects are groupable.

Complex Reorganization

In complex reorganization, objects or groups are broken down into their parts, compared, and then grouped back together, so as to produce a more similar set of elements in the two images.

For example, consider Figure 3.22. 22A contains two objects, a square and an 'X.' 22B contains one object, an hourglass. In the initial image comparison, the square maps to the hourglass. The objects are different, but neither is a subshape of the other. Thus, basic reorganization is impossible. However, the objects do contain some similar elements, e.g., edges with similar lengths and orientations. In such cases, the model can attempt complex perceptual reorganization. The process requires two steps.

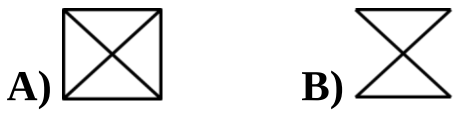


Figure 3.22 Complex reorganization facilitates this comparison.

1) Break down the objects into their edges, such that every edge is now a separate object, and repeat the comparison. Here, the square's top and bottom edges will map to the hourglass's top and bottom edges. The 'X' within the square will map to one of the hourglass's diagonals, causing another shape mismatch. Repeat step 1), breaking the 'X' shape down into its two edges.

2) Identify a set of edge objects that correspond between the images. If these edge objects are connected, and if they form identical objects in the two images, join them back together in each image. Here, the top, bottom, and diagonal edges align between the images. These get grouped together to form an hourglass in each image. Repeat the comparison.

This results in an hourglass object for both images. At this point, 22A contains three objects: the hourglass and two vertical edges. 22B contains only one object: the hourglass. The difference between the images, then, is that the two vertical edges in 22A are removed from the hourglass in 22B.

Complex perceptual reorganization typically requires more comparisons than basic reorganization. Given its complexity, it may not be an automatic process. That is, a typical person might not look at Figure 3.22 and see two hourglasses. I will return to this question later.

4. Spatial Routines for Sketches

Spatial Routines for Sketches (SRS) is based on Ullman's (1987) *visual routines* model. Ullman suggested that people have several basic, low-level visual operations. These include tracing along a curve and coloring in a surface. This finite set of operations can be combined in different ways to create a near-infinite variety of visual routines. Routines are analogous to computer programs: operations are executed sequentially, with the output of each operation providing the input for future operations. Each routine represents a strategy for computing some high-level visual feature, such as a qualitative spatial relation.

An advantage of visual routines is that they allow researchers to experiment with different encoding strategies. For example, suppose I have a theory for how people determine positional relations (e.g., **rightOf**). If I can write a routine based on that theory, then I know the theory is feasible—it can be implemented using basic operations available to people, so a person *could* use my approach. If I have a computational system for executing the operations, then I can test how my routine performs on actual stimuli.

Whereas visual routines model visual encoding, spatial routines model spatial problem-solving. SRS's operations are much higher-level. For example, visual perception is a single operation. Image comparison and shape comparison are also operations. As in many programming languages, operations can call each other. For example, image comparison can call the shape comparison operation.

SRS implements several additional features to support flexible problem-solving. Firstly, operations are highly parameterizable. Consider visual perception, which builds a qualitative image representation. A spatial routine can specify the level of abstraction (edges, objects, or groups) and whether the representation should be orientation-specific. If desired, the routine can also specify many smaller details. For example, should space objects be detected, or should relative size be encoded? Thus, an operation can be parameterized to produce the desired output for a particular routine.

Secondly, many of the control structures common to programming languages are built into SRS. These include if-then statements and loops (for-loops and repeats). These structures allow routines to utilize multiple strategies for solving a problem. A routine can attempt one approach, evaluate the solution, and backtrack if necessary. Thus, SRS supports the control processes that are necessary for problem-solving.

Finally, operations can be updated. For example, suppose a routine discovers there is a problem with its image representations and backtracks. Rather than repeating the perception process from scratch, the routine can simply specify what needs to change in an image representation. This includes changes to the parameters, such as moving to a different level of abstraction.

Updates can also include *directives*. Directives are commands to change some aspect of an operation. Whereas parameters are general to an operation type (e.g., visual perception), directives are specific to an operation instance. For example, suppose visual perception identifies two objects in Figure 4.1: a square and an 'X.' A directive might say to break the square down into its four edges, treating them as separate objects, and generate a new image representation. A later directive might say to group the 'X' shape with top and bottom edges. After the visual perception operation was updated twice, with these directives, it would represent the image as an hourglass touching two vertical lines.

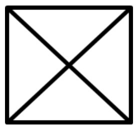


Figure 4.1 An ambiguous image.

In the following section, I discuss the different categories of operations available in SRS. I then list the operations, providing common parameters and directives for each. After this, I describe the programming language for writing spatial routines and provide a simple example routine for solving Bongard problems (Bongard, 1970). Finally, I outline SRS's data-gathering capabilities.

4.1 Operation Categories

I believe much of the work in spatial problem-solving is done by three categories of operations: perception, comparison, and inference. I have already discussed perception in detail. Below, I describe the other two.

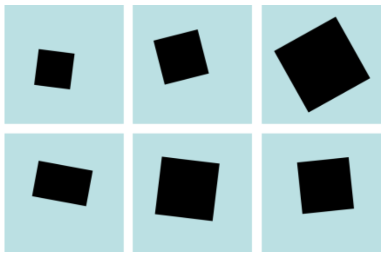
Comparison

Comparison involves aligning two representations using the Structure-Mapping Engine (SME). Thus far, I have described two types of comparison: image comparison, and shape comparison. SRS has an operation for each of these. Image comparison can be basic or analytic. Basic comparison involves a single SME mapping, but in analytic comparison, the corresponding objects are compared via vhape comparison.

SRS also has larger-scale operations that perform comparison for different purposes. Recall that SME comparisons can identify both commonalities and differences. These commonalities and differences are each important during problem-solving.

For example, commonalities are needed for the oddity task (Figure 4.2A). To determine which image is different, one must first determine what is common across the images. In Figure 4.2A, a comparison of the images in the first row indicates that they all contain straight, closed, fully-symmetric shapes. In contrast, the rectangle in the lower left image is not fully symmetric.

On the other hand, differences are generally needed for Raven's Progressive Matrices (Figure 4.2B). To understand a matrix, one must determine what is changing between images (Carpenter, Just, & Shell, 1990). In Figure 4.2B, a comparison of the images in the top row indicates that one object is rotating clockwise as it moves to the right, relative to the other object. Applying this to the bottom row, one expects to see a trapezoid with a horizontal rectangle on its right (answer 4).

A) 

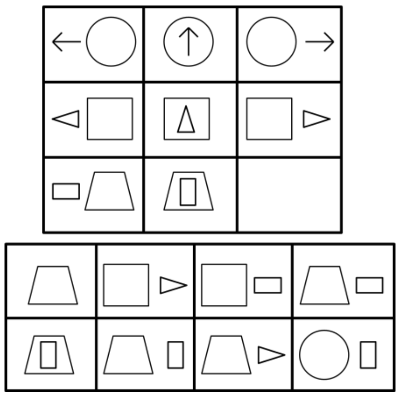
B) 

Figure 4.2 A: Oddity task problem from (Dahaene et al., 2006). B: Raven's Matrix-type problem.

In each of the above examples, more than two images are compared. Thus, an Image Comparison operation is insufficient. We need an operation that can compare multiple items and determine their commonalities or differences. Note that the items being compared are not always images. For example, suppose one compares the images in the top two rows of Figure 4.2B, determining the differences in each row. One might then perform a *second-order comparison* to identify the commonalities in the two sets of differences. This would produce a generalization describing what changes across each row of the matrix.

SME produces both commonalities and differences whenever it performs a comparison. Thus, one might expect that a single operation could handle comparing to find commonalities and comparing to find differences. However, there are key distinctions between these comparisons when more than two items are involved.

The operation for finding commonalities uses *analogical abstraction* (Kuehne et al., 2000) to construct a generalization containing what's true in all the items being compared. For example, consider the top row in Figure 4.2A. It begins by comparing the first two images. Any expressions in the representations that fail to align are discarded, leaving a generalization containing expressions that are true about both images. Then, it compares this generalization to the third image. Again, unaligned expressions are discarded. For example, because relative size varies across the images, this is lost. The resulting generalization describes an object that is closed, straight, and fully symmetric, with a black border and a black fill.

In contrast, the differences between several items cannot be represented with a single generalization. Instead, difference-finding identifies the differences between each consecutive pair of items. For example, consider the top row in Figure 4.2B. Comparing the first two images, the model determines that the arrow rotates 90° and moves from being left of the circle to being within the circle. In the next two images, the arrow rotates 90° and moves from being within the circle to being right of the circle. The resulting representation describes how things change between each pair of images.

The differences between generalization and difference-finding have important ramifications for image comparison. During *difference-finding* on images, analytic image comparison is quite helpful. For Figure 4.2B, it allows the model to see that the arrow is rotating. However, there are two reasons that *generalization* wouldn't use analytic image comparison:

1) The objective is to represent what's common across images, whereas shape comparison identifies differences, i.e., the shape transformations.

2) Analytic image comparison only works between two image representations, since it compares their corresponding objects' shapes. Here, a generalization (say, of the first two images in Figure 4.2A) will be compared to a single image. Since the generalization represents multiple images, there are no concrete objects available for shape comparison.

Thus, generalization uses basic image comparison, whereas difference-finding uses analytic image comparison. Of course, this only applies if the items being compared are images. Both operations can work on any representation, including the output of other generalization or difference-finding operations.

Inference

Some researchers (Sternberg, 1977; Bethell-Fox, Lohman, & Snow, 1984) believe geometric analogy involves: 1) comparing images A and B to find the differences between them; 2) comparing A and C to find the corresponding elements; and 3) applying the A/B differences to the corresponding elements in C to produce D’, a proposed solution. I shall refer to steps 2) and 3) as *visual inference*.

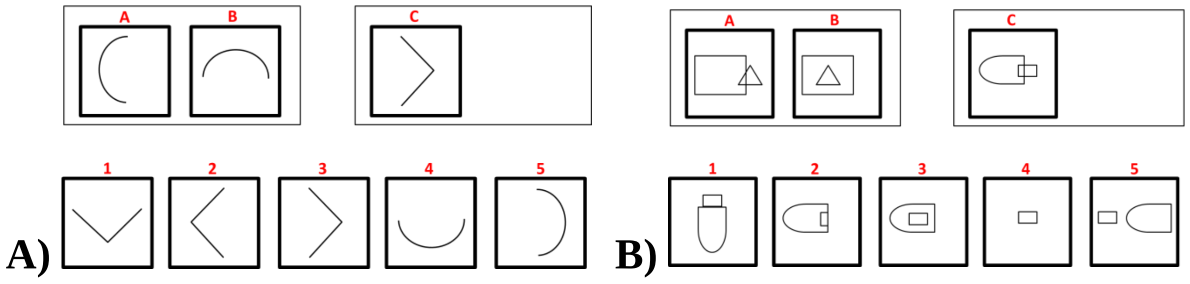


Figure 4.3 Geometric analogy problems.

Like visual comparison, visual inference operates at multiple levels of the hierarchy. In shape inference, one takes a shape transformation (e.g., the 90° rotation in Figure 4.3A) and applies it to an object (the corner in image C) to produce a novel object. Thus, shape inference is another form of spatial visualization. It is also a prime example of mental imagery (Kosslyn, 1980), as it generates a novel spatial entity.

In image inference, one takes image differences (e.g., between A and B in Figure 4.3B) and applies them to another image (C) to produce a novel image representation (D’). This requires three steps:

1) Compare image A to image C to find the corresponding elements.

2) Take all the spatial relation changes in the A/B differences. Apply them to the corresponding elements in C to identify the spatial relations that should hold in D’.

3) Take all the shape transformations in A/B. Apply them to the elements in C, using shape inference. Generate new shape attributes based on the resulting object shapes.

Note that visual inference is never required for solving a geometric analogy problem. An alternate approach (Evans, 1968; Mulholland, Pellegrino, & Glaser, 1980) is to: 1) compare images A and B to identify differences between them; 2) for each possible answer, compare C to that answer to identify the differences between them; 3) compare the A/B differences to the C/answer differences, and pick the answer that produces the most similar differences. By incorporating both flexible visual comparison and visual inference, spatial routines can use either approach to solve geometric analogy and Raven's Progressive Matrix problems. I compare these two strategies in later chapters.

4.2 Spatial Operations

I now describe SRS's spatial operations. Each operation's name is an active verb. For example, *Perceive* constructs image representations, and *Compare-Images* compares them.

Each operation takes a list of inputs, a set of parameters, and a set of directives. Inputs are the elements over which one is operating. Frequently, they are the outputs of other operations. For example, *Generalize* might operate over several image representations computed by *Perceive*. Alternatively, if an operation is to be updated with new parameters or directives, then the input is simply the result of the previous operation call.

Parameters come in (type value) pairs. For example, (**:focus :edges**) means the representation should be encoded at the edge level of abstraction. A given operation, such as *Perceive*, has many parameters. However, each has a default value. The spatial routine can leave any or all parameters undefined, and these parameters will take on their default values.

Recall that operations sometimes call other operations. For example, *Find-Differences* might call *Compare-Images*, which might call *Compare-Shapes*. In a *Find-Differences* operation call, a routine can specify parameters for any *Compare-Shapes* operations that get called. This is done with another (type value) pair, but now the type is the operation type, and value is the list of parameters. For example (**:compare-shapes** (**:top-transform :rotation**)) means that if *Compare-Shapes* is called, its **:top-transform** parameter should have the value **:rotation**—this means that it prefers to find rotations between shapes, instead of preferring perfect identity matches.

Directives also come in (type value) pairs. However, the values refer to specific elements in the operation, e.g., objects in an image representation. For example, directives might specify that particularly objects should be broken down or grouped together in a *Perceive* update. Unlike parameters, directives have no default values. Directives are only applied if a value is specified.

Many operations produce a data structure as output. The nature of this data structure depends on the operation. Often, it will include a qualitative representation. For example, *Perceive* produces an image representation, while *Generalize* produces a generalization over several representations. Qualitative representations are lists of predicate calculus expressions describing entities, attributes of entities, and relations between entities. These representations can serve as the input for later *Generalize* or *Find-Difference* operations.

An operation's output data structures may also include a results list. Again, this is a list of (type value) pairs specific to the type of operation. For example, the results list for *Perceive* includes **:num-elements**, which indicates the number of elements in the image representation. As we shall see later, spatial routines can use these results to evaluate an operation and control the sequence of future operations—for example, if-then statements can trigger off particular result values. To save processing time, some complex results are only computed on demand, if the routine queries for that result from a particular operation.

Finally, many operation data structures include a score. The meaning of the score, again, depends on the operation type, but typically it is a similarity measure computed by SME. For example, in *Compare-Images*, it is the similarity of the two images. In *Generalize,* it is the lowest similarity score from the sequence of comparisons. Again, routines can use the score to evaluate an operation.

Below, I list the spatial operations. For each operation, I describe the inputs, common parameters and directives, results, and score.

Locate

Before images can be encoded, they must be located in the problem. Usually, images are located based on *sketch lattices,* grids that can be generated in CogSketch. For example, geometric analogy problems require three sketch lattices (Figure 4.4): one displaying images A and B, one displaying image C, and one displaying the possible answer images. It is the user's responsibility to move each image's objects into the appropriate cell of the appropriate sketch lattice. Once this is done, spatial routines can find an image in the lattice using the *Locate* operation.

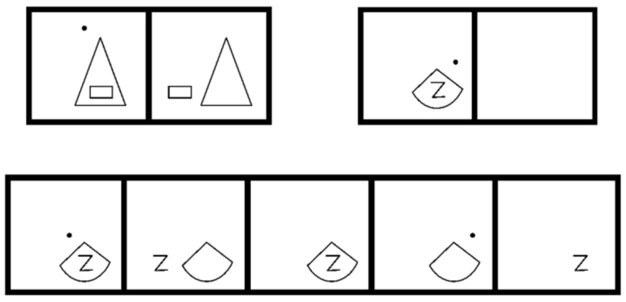


Figure 4.4 A geometric analogy problem, as it looks in CogSketch.

Input

The input to *Locate* is a list of keywords that provide progressively more specific information. For example, the list (**:x** 1 **: y** 1) indicates the leftmost and topmost sketch lattice, while the list (**:y** 2) indicates the lower sketch lattice—in Figure 4.4, the **:x** value needn't be supplied, as there is only one lattice in the second row. The list (**:x** 1 **:y** 1 **:end :x** 1) indicates the leftmost image in the leftmost and topmost sketch lattice, i.e., image A. One can either specify this entire list in a single *Locate* or perform two *Locate* operations, one to find the sketch lattice and one to find the image in the sketch lattice.

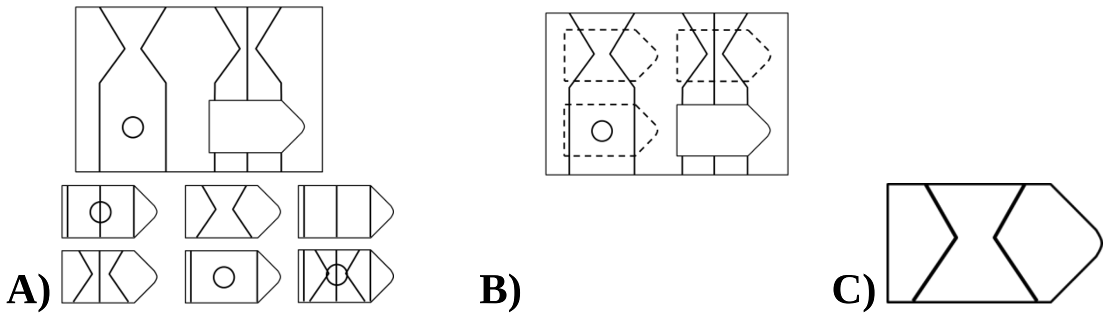


Figure 4.5 A Raven's Progressive Matrices 'A' problem (not from the actual test).

Sometimes, locating images is more difficult. Figure 4.5A shows a facsimile of a problem from the Standard Progressive Matrices, section A. Rather than containing a matrix, these problems contain an image with a piece missing. The test-taker must choose the answer that best completes the image. To simulate the information given to the test-taker, we manually name the large rectangle surrounding the whole image "Problem," and the smaller object surrounding the missing piece "Answer"—naming objects is a simple step in CogSketch.

The spatial routine solves this problem by treating the missing piece as the lower-right image in a 2x2 matrix and locating the other three entries in the matrix (Figure 4.5B), thus producing a typical matrix problem. The input to find the upper left image would look like this: ("Problem" **:x** 1 **:y** 1 **:referent** "Answer" **:x** 2 **:y** 2). This translates as "Take an image with everything inside the object named 'Problem.' Within that image, take the upper left location. As a referent, the object named 'Answer' is the lower right location." To create this location, *Locate* draws out the boundary (as in Figure 4.5B), and intersects it with the objects lying inside "Problem." It creates new objects for any object parts that lie within the boundary, thus producing a new image (Figure 4.5C).

Parameters

:fully-specified-only? Indicates whether the operation should only return actual image locations. For example, suppose the inputs refer to a sketch lattice, but not to a specific image within the lattice. If this parameter is *false*, the operation will simply return the location of the overall lattice. If the parameter is *true*, the operation will return a list of locations for each image in the lattice.

Output

The output describes a particular location in the problem (a sketch lattice, one row of a sketch lattice, an image in a sketch lattice, the area inside a particular object, etc). The location can serve as the first item in a list of inputs for a future *Locate* operation. It can also serve as the input to a *Perceive* operation.

Results

:num-rows/:num-cols The number of rows or columns in a located sketch lattice.

:num-elements The number of elements in a located image.

Perceive

The *Perceive* operation generates a qualitative spatial representation for an image. It performs both perceptual organization and perceptual encoding. Thus, given an object, it can generate an edge-level representation, and given several objects, it can generate a group-level representation.

Inputs

The input can be a location. Alternatively, it can be a list of *Locate* keywords, in which case *Perceive* resolves the location in the same way that *Locate* does. If a location refers to several images (e.g., a sketch lattice, or one row in a lattice), then *Perceive* will return a list of image representations.

Parameters

The following is a representative sample of common parameters. Overall, *Perceive* has a large number of parameters, allowing the user to heavily customize what gets represented for an image.

:focus Level of abstraction for the representation. The value can be **:groups**, **:objects**, or **:edges**.

:orientation-specific? Should the representation be orientation-specific?

:size-specific? Should the representation include terms relating to relative size?

:textures? Should textures be computed?

:shape-trans-relations? Should shape transformation relations (i.e., rotations and reflections) be computed?

:shape-features A list of keywords specifying shape features that should be included (e.g., **:symmetry**).

Directives

Recall that directives specify changes that should be made to specific elements when the operation is updated.

:segment-elements These objects should be broken down into their edges, such as that each edge is a separate object; or these groups should be broken down into their objects. This supports complex perceptual reorganization (see section 3.5) or group comparison.

:group-elements These elements should be joined together to form a single object or group. This also supports complex perceptual reorganization.

:segment-elements-as-directed These objects or groups should be broken down such that a specified subset of their parts remain together. Usually, this means splitting one object or group into two. This supports basic perceptual reorganization.

:mark-segmented-elements? If this is *true*, then when an element is segmented into its parts, mark each as important. This is used in group comparison. Recall that the model compares two groups by breaking them down into their objects and comparing the overall images, while requiring that the objects from the groups (which are marked as important) be matched to each other.

Output

Perceive produces a qualitative image representation. An image representation contains:

1) A list of predicate calculus expressions describing attributes and relations for the objects in the image.

2) The objects (or groups) themselves. Each object can be represented both qualitative and quantitatively at the edge level, supporting shape comparison (see section 3.4).

This representation can serve as the input to future comparison operations. If *Perceive* is called on a list of locations, it will return a list of image representations.

If *Perceive* was forced to segment or group elements due to a directive, then its **:focus** parameter no longer truthfully reflects the representation's focus. In this case, the operation's parameter is changed to **:dynamic**. This is important because spatial routine code can be conditional on a previous operation's parameter values.

Results

:num-elements The number of elements in the representation.

:closed-shapes? Are any of the objects closed shapes?

:new-groupings Any new groups created due to a **:group-elements** directive.

Compare-Shapes

This operation compares two shapes, using the algorithm from the last chapter. It works on either objects or groups. It attempts to find the simplest possible transformation between the shapes. While a spatial routine can call this operation directly, *Compare-Shapes* is usually called by *Perceive* or *Compare-Images*.

Input

Compare-Shapes takes two objects or groups.

Parameters

:transformations Specifies the transformations to try, from simplest to most complex. The default list is (**:identity :regular-reflection :rotation :reflection :deformation**). Note that **:regular-reflection** refers to reflection across a canonical axis.

:top-transform Provides an alternate means of changing the ordering. A routine can specify that one transform type (e.g., **:rotation**) should be tried first.

:reflection-in-space? Are reflections spatial transformations? I.e., should the axis of reflection lie between the two objects (see section 3.4)?

:deformations The list of deformation types to try.

Output

Compare-Shapes returns a data structure describing the quantitative transformation between the two objects. This might be a rotation angle or an axis of reflection. Recall, however, that for deformations, there is a separate quantitative transformation for each edge, since edges may vary in how their length or position changes. There is also a scale value: the change in size between the objects. Usually, this is the scale change for the overall objects. However, for some deformations (e.g., **DeformedShape-Longer**), the scale has a more specific meaning.

While this data structure contains the *quantitative* transformation, the *qualitative* transformation is given in the results below.

Results

:transform-value A qualitative label for the transformation (see Table 3.5). For example, **ReflectedShape-XAxis**, or **DeformedShape-AddedPart**.

:scale-value A qualitative label for the change in scale, if there is a change (**ScaledShape-Larger** or **ScaledShape-Smaller**).

:subshape-deformation Gives the corresponding parts if one object or group is a subshape of the other. If so, basic perceptual reorganization may be possible. This result can serve as the **:segment-elements-as-directed** directive for a *Perceive* update.

:some-parts-overlap? Indicates whether the two elements have some common parts. If so, complex perceptual reorganization may be possible.

Compare-Images

This operation compares two images by aligning their qualitative representations. In a typical image comparison, any element is free to map to any other element. However, *Compare-Images* can also model a more conservative comparison in which elements must match to other elements with the same shape. To do this, the model registers each element in the Shape Registry beforehand. Then, image representations are supplemented with attributes for each element’s shape class. During comparison, SME mapping constraints specify that elements must match to other elements with the same shape class attribute.

Input

Compare-Images takes two image representations.

Parameters

:analytic? Is this analytic or basic image comparison?

:shape-constrained? Only allow elements to align if they're the same shape class?

:strict-shape-constrained? Only allow elements to align if they're the same strict shape class?

:mark-constrained? If an element is marked as important, can it only map to another important element?

:ignore-shape-features? Abstract out shape features when comparing?

:ignore-spatial-relations? Abstract out spatial relations and relative location attributes when comparing?

:coverage Determines how the comparison is scored. See below.

Directives

:different-mapping-from Try to find a mapping with different correspondences than this one. Recall that SME finds up to three global mappings. Normally, the operation returns the highest-scoring mapping, but in this case, it will return the highest-scoring mapping that has different correspondences. This is an important form of backtracking, when the initial mapping proves problematic.

:constraining-shape-types Constrain the mapping for these shape types only.

:constraining-strict-shape-types Constrain the mapping for these strict shape types only.

Output

Compare-Images produces an SME mapping between the two images. The mapping includes commonalities and differences. If this is an analytic image comparison, there will also be differences based on the shape comparisons.

Score

This is the similarity of the two images, according to the SME mapping score. Because mapping scores depend on the sizes of the representations, the score is normalized based on their self-mapping scores. A representation's self-mapping score is the score it would get when compared to itself; it is the maximum possible score for a mapping with that representation.

The type of normalization depends on the **:coverage** parameter. **:base** coverage indicates one is measuring how well the mapping covers the base representation, i.e., the first of the two images. Thus, the score is divided by the base's self-mapping score. **:symmetric** coverage indicates one is measuring how well the mapping covers both representations. Thus, the score is divided by the average of the two representations' self-mapping scores.

If this is an analytic image comparison, the score will be updated to reflect whether corresponding objects are identical shapes, rather than being based on the breadth of aligned shape attributes.

Results

:differences-detected? Indicates whether any differences were detected between the representations. There are three ways to detect differences: 1) from candidate inferences in the mapping; 2) from elements in one representation that did not map to anything in the other; or 3) from shape transformations.

Generalize

This operation constructs a generalization via analogical abstraction (Kuehne et al., 2000). Given a list of representations, the operation compares the first two, abstracting out any expressions that failed to align, to produce a generalization. It then compares this generalization to the next representation, again abstracting out whatever does not align. This continues until the generalization contains only what is true across all the representations.

Input

The input is a list of qualitative representations; these might be image representations, the results of other *Generalize* operations, or the results of *Find-Differences* operations (see below). During generalization, the operation calls *Compare-Images* any time two image representations are compared. Otherwise, it simply makes an SME comparison.

Parameters

:coverage Determines how comparisons are scored. See above.

Output

Generalize produces a generalization, i.e., a qualitative representation describing the commonalities across its inputs. This generalization can serve as the input to future comparisons. For example, a generalization across several images might later be compared to a single image.

Score

The generalization's score is the minimum mapping score, across all the comparisons that produced the generalization.

Results

:differences-detected? Were any differences detected during the comparisons?

Find-Differences

This operation takes a list of representations and finds the differences between each consecutive pair. It produces a *pattern of variance,* describing how the elements change from one representation to the next. For example, consider Figure 4.6. *Find-Differences* would compare the first two and the last two images. Based on the mapping correspondences, it would identify three elements in this row: the circle, the arrow, and the dot. It would then represent how these three elements change between the images.

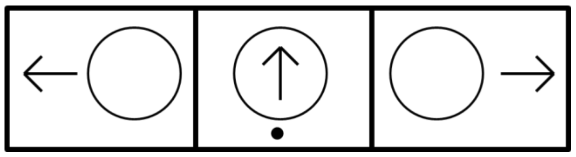


Figure 4.6 A row of images for difference-finding.

((**ElementInARowFn** 0) Image-1) ((**ElementInARowFn** 1) Image-2)	(**changeBetweenImages** (**elementContains** Circle Arrow) Image-1 Image-2) (**changeBetweenImages** (**centeredOn** Arrow Circle) Image-1 Image-2) (**changeBetweenImages** (**rightOf** Circle Arrow) Image-2 Image-1)) (**changeBetweenImages** (**above** Circle Dot) Image-1 Image-2))
(**changeBetweenImagesFromTo** (**RotatedShape-90** Arrow) (**RotatedShape-90** Arrow) Image-1 Image-2) (**changeBetweenImagesFromTo** (**RemovedShape** Dot) ((**AddedShapeTypeFn** (**StrictShapeTypeFn** 2 0)) Dot) Image-1 Image-2)	

Table 4.1 Pattern of variance for the first two images in Figure 4.6[4].

Differences are identified in three ways (see Table 4.1[4] for the differences between the first two images in Figure 4.6). Firstly, they are identified by candidate inferences in the SME comparisons. For

---

[4] For simplicity, elements are titled by their shape, e.g., “Circle.” Actually, elements receive arbitrary names like “Element-1.” In addition, *Find-Differences* wraps an expression around each element to specify that it is the instance of that element from a particular image. For example, the full form of the bottom right expression in Table 4.1 is:

(changeBetweenImages

(above (ElementInImageFn Element-2 Image-2) (ElementAddedInImageFn Element-3 Image-2))

Image-1 Image-2)

example, in the first image, the arrow is right of the circle, but in the second image, the circle contains the arrow, and the arrow is centered on the circle. These differences are usually represented with the **changeBetweenImages** predicate (Table 4.1, second column). However, when two elements switch places in a relation, e.g., the change is from (**rightOf** Circle Arrow) to (**rightOf** Arrow Circle), the difference is represented with **reversalBetweenImages**.

The other differences can only be detected between image representations. If there is a shape transformation, or if one object changes to an entirely different shape, this is represented with **changeBetweenImagesFromTo**. Table 4.1 gives an example for the rotating arrow shape. In this case, the two transformations listed are both **RotatedShape-90**, as the inverse of a 90° rotation is a 90° rotation. Note that if an object's shape is identical in every image, no shape changes are represented.

Finally, if an object is added or removed between the images, this is also represented with **changeBetweenImagesFromTo** (Table 4.1, bottom of left column). Note that *Find-Differences* explicitly encodes the strict shape class of the added object. This is equivalent to remembering "a dot shape is added between these images."

This approach loses one piece of information. Because analytic image comparison filters out relative location attributes (e.g., **OnTop-Element**), patterns of variance do not describe changes in relative location. However, relative positional relations (such as **rightOf**) provide more useful information about changes to an object's location.

The above describes the default format for a pattern of variance. However, some Raven's Progressive Matrix problems require strategic shifts to this format. Two parameters can change how the pattern of variance is represented. Note that, while *Find-Differences* can operate over any representation, the special formats below are only meaningful when it is applies to image representations.

:pov-type

The pov-type can be either **:diffs** or **:literal**. **:diffs** indicates one is representing the differences between images. The above representation scheme is used. Additionally, when two patterns of variance are compared, strict ordering constraints are applied to the SME mapping. This means that the first image maps to the first image, the second image maps to the second, etc.

:literal indicates one is representing what is literally true in each image. To construct this, *Find-Differences* first compares the images and detects differences between them. It then constructs a representation for each image, filtering out any expressions that hold for all the images. As with analytic image comparison, shape features are abstracted out. Instead, each object gets an attribute describing its strict shape class.

For example, consider the top row in Figure 4.7A. Every image contains a smaller black circle, so this information is abstracted out. The resulting pattern of variance describes the strict shape class for the outer shape in each image. Essentially, the representation says, “One image has a square, one has a circle, and one has a diamond.” Literal patterns of variance do not use strict ordering constraints. Thus, the top row maps perfectly to the second row; they each contain a circle, a diamond, and a square, but in different orders.

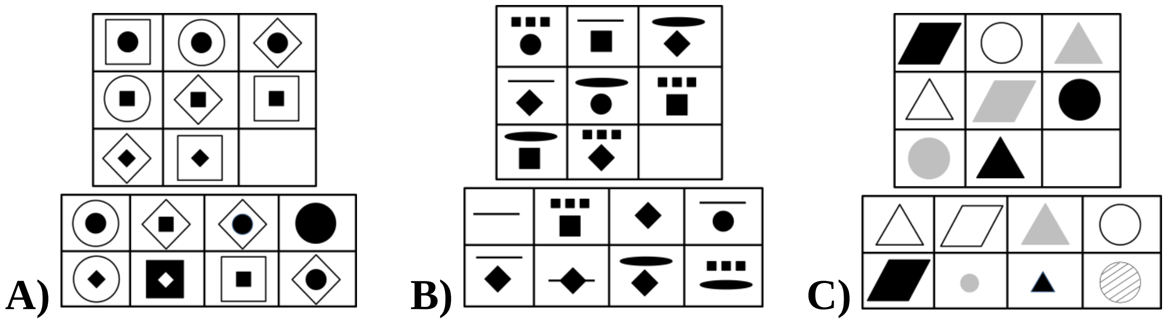


Figure 4.7 Raven’s Progressive Matrix problems requiring a literal pattern of variance.

:simple?

If a pattern of variance is *simple*, then it does not represent differences between images. Instead, it breaks each image up into its objects and represents differences between individual objects. For example, Figure 4.7B shows a problem that would require a simple, literal pattern of variance. Each row contains a square, a circle, and a diamond; and a group, a line, and an ellipse. However, the two sets of objects combine differently in each row. Thus, one must think about the objects without binding them together in images.

Alternatively, if every image contains only a single object, then a simple, literal pattern of variance breaks the objects down into their features and represents those separately. In Figure 4.7C, each row contains a parallelogram, a circle, and a triangle; and a black object, a white object, and a gray object.

A simple, difference pattern of variance is similar: one represents how each object changes between images, without tying objects together in an image.

A simple pattern of variance is the most abstract kind, since it abstracts out the image itself. Two things necessarily following from this: 1) strict ordering constraints are not applied, as there are no images to apply them to; 2) spatial relations are not represented, as objects are no longer tied together in an image.

Input

Find-Differences takes a list of representations. Often, this is a list of images, and indeed some aspects of *Find-Differences* (e.g., analytic image comparison) only work for images. However, the operation can handle any other representation, including generalizations or other patterns of variance.

Parameters

:pov-type Determines how patterns of variance are represented. See above.

:simple? Determines how patterns of variance are represented. See above.

:first-to-last? Compare the first item to the last and include those differences in the representation?

:analytic? Use analytic image comparison? The default is *true*.

:shape-transformations? Encode shape transformations? The default is *true*.

:always-encode-shapes-constant? Encode a **ConstantShape** transformation between identical objects, even when the object is identical across all images? If this is *false*, *Find-Differences* will still encode a **ConstantShape** transformation if the object is identical across some images but different across others (e.g., it's in the first two images but not in the last). The default is *false*.

Directives

Finding an appropriate pattern of variance can be a key step during problem-solving. Thus, there are several directives that can change the pattern of variance. In many cases, a result from one *Find-Differences* call can serve as a directive in the next.

The first three directives involve changes to a pattern of variance's input image representations, and thus they are passed on to the *Perceive* operation.

:segment-elements Update the input image representations, applying this directive to their *Perceive* operations.

:segment-elements-as-directed Update the input image representations, applying this directive to their *Perceive* operations.

:new-groups Update the input image representations. This directive includes **:group-elements** directives for each image.

The remaining directives involve changes to the SME comparisons between representations.

:try-different-mappings? Find alternate mappings with a different set of corresponding elements between the inputs.

:preferred-mappings Here is a set of mappings we'd like to achieve. Find alternate mappings with the corresponding elements consistent with these preferred mappings.

:constraining-shape-types Apply this to the image comparisons.

:constraining-strict-shape-types Apply this to image comparisons.

:keep-old-constraints? Keep any of the above constraining shape types from the previous pattern of variance when updating.

Output

Find-Differences generates a pattern of variance, a representation of the differences between images. Again, this is a list of predicate calculus expressions that can serve as the input for future comparisons.

Score

The score is the minimum mapping score, across all the comparisons that produced the pattern of variance.

Results

In many cases, a result can serve as the input to a future directive.

:differences-detected? Were any differences detected during the comparisons?

:attributes-only? Is this a simple pattern of variance in which we broke objects down into their features and represented each separately (e.g., Figure 4.7C)?

:subshape-divided-shapes A list of subshape deformations between objects. Can serve as input to the **:segment-elements-as-directed** directive, thus supporting basic perceptual reorganization (see section 3.5).

:partially-matched-shapes A list of objects that have some common parts. Can serve as input to the **:segment-elements** directive, thus supporting complex perceptual reorganization.

:first-to-last-match-objects When the **:first-to-last?** parameter is *false* but there are objects in the first or last image that fail to match well (either they map to nothing or they map to something with an entirely different shape), *Find-Differences* compare the first and last images. If it finds identical shape matches for these objects, it lists them here. For example, in Figure 4.8, the vertical line in the first image matches the vertical line in the last image. This will often trigger an update with **:first-to-last?** set to *true*. See Chapter 7, describing the Raven's Progressive Matrices model, for more details.

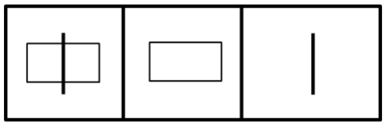


Figure 4.8 A row of images with first-to-last matches.

:first-to-last-match-shapes/:first-to-last-match-strict-shapes Shape classes or strict shape classes for first-to-latch-match objects. Can serve as input to **:constraining-shape-types/:constraining-strict-shape-types**.

:all-perfect-match-shapes/:all-perfect-match-strict-shapes Shape classes or strict shape classes for all corresponding objects with perfect shape matches, including first-to-last-match objects. Can serve as input to **:constraining-shape-types/:constraining-strict-shape-types**.

:elements-broken-down? Have any elements been broken down because of a **:segment-elements** or **:segment-elements-as-directed** directive?

:groupable-shapes List of objects that could be grouped back up for each input image. Can serve as input to **:new-groups**. Supports the last step of complex perceptual reorganization.

:mismatched-objects List of all objects that map to other objects with an entirely different shape (no valid transformation or deformation).

:better-mappings-for-mismatched-objects Some alternate SME mappings that would result in fewer mismatched objects and more perfect identity matches. Recall that SME returns up to three mappings for a typical comparison. Thus, a better mapping could be the second or third from the SME comparison. Can serve as input to **:preferred-mappings**.

Sometimes, *Find-Differences* is useful for determining whether there are differences of a certain type between items. For example, consider Figure 4.9A. In this oddity task problem, the orientation of the lines changes between each image. Thus, when picking the odd image out, one knows that orientation is unlikely to be a factor.

What if there are differences, but only because of the odd image out? In Figure 4.9B, the only image with a different orientation is the answer. To check for this, we can look for *nonadjacent differences*. That is, if we combine the two rows into one list of images and *Find-Differences* across them, are there orientation differences in two nonadjacent comparisons? If so, those differences can't be linked to a single image.

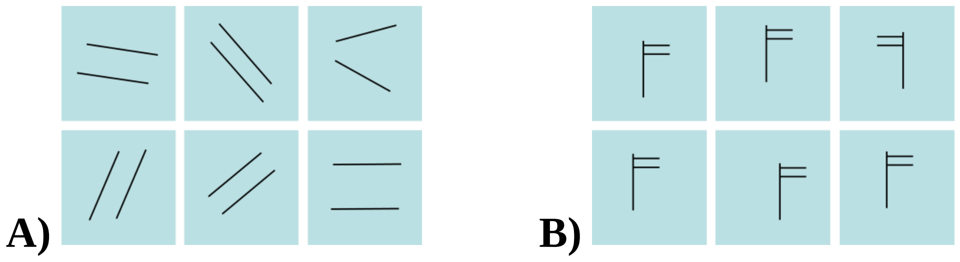


Figure 4.9 Oddity task problems with orientation differences.

:orientation-differences? Were there differences in orientation, i.e., positional relation changes, rotations, or reflections?

:shape-differences? Were there any changes to corresponding objects' shapes (deformations, complete shape changes)?

:size-differences? Were there differences in the sizes of corresponding objects?

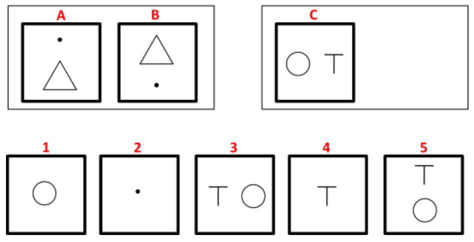
:non-adjacent-orientation-differences?/:non-adjacent-shape-differences?/:non-adjacent-size-differences? Were there non adjacent differences of these types?

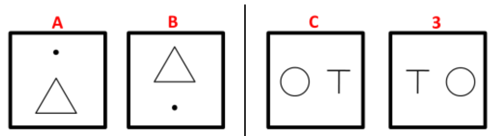
(:non-adjacent-differences-of-types :orientation :shape :size) Were there nonadjacent differences of the listed types (some subset of (**:orientation :shape :size**))?

Comparing Patterns of Variance

Because patterns of variance represent differences between items, they are more abstract than image representations or generalizations. Thus, they can be compared more flexibly by other operations. This flexible comparison deserves a more detailed explanation.

Consider the geometric analogy problem below (Figure 4.10). The difference between A and B is a reversal in the **above** relation, while the difference between C and 3, the correct answer, is a reversal in the **rightOf** relation. These are analogous differences—in each case, a reversal in a positional relation causes the objects to switch places. Thus, it is useful for pattern of variance comparisons to align these differences.

A)


B)


(**reversalBetweenImages** (**above** Dot Triangle) (**above** Triangle Dot) Image-1 Image-2)	(**reversalBetweenImages** (**rightOf** T Circle) (**rightOf** Circle T) Image-1 Image-2)

Figure 4.10 A: A geometric analogy problem. B: The patterns of variance.

Fortunately, SME has built-in support for aligning analogous expressions like the above. Recall that normally, only identical predicates in SME can align. *Tiered identicality* (Falkenhainer, 1990) allows nonidentical predicates to align when: a) they are the arguments of aligned expressions; and b) the predicates are semantically similar.

Here, the first criterion is met because the **above** and **rightOf** expressions are both arguments of **reversalBetweenImages** expressions (Figure 4.10B). Similarly, nonidentical predicates can align when they are the arguments of **changeBetweenImages** or **changeBetweenImagesFromTo** expressions. This explains why tiered identicality doesn't work for basic image comparison, or comparison of image generalizations—there is never a common higher-order predicate like **reversalBetweenImages** to put **rightOf** and **above** into correspondence.

The second criterion depends upon the knowledge base. Recall that CogSketch utilizes a knowledge base with 58,000 concepts. This knowledge base defines an ontology in which, for any given term, there may be more specific and more general forms. I have supplemented this ontology with categories for the spatial terms in my qualitative vocabulary. For example, **above** and **rightOf** share a more general predicate: **positional-Generic**. Because they share this predicate, SME knows that they are semantically similar.

The above approach to similarity is termed *minimal ascension* (Falkenhainer, 1990). That is, if we move upward (towards the more general) in the ontology of terms, what is the minimum number of levels we must ascend to find a common term. In my experiments, I use a max minimal ascension depth of 1. Thus, terms must share a common immediate parent in the ontology.

Terms from Object-Level Image Representation (Table 3.3)	
positional-Generic	**above, below**
rotatedShapes	**rotatedShapes-90, rotatedShapes-180**
reflectedShapes	**reflectedShapes-XAxis, reflectedShapes-YAxis**
AtLocation-Element	**OnTop-Element, OnLeft-Element**
Terms from Shape Transformations Between Images (Table 3.5)	
RotatedShape	**RotatedShape-45, RotatedShape-90, RotatedShape-135, RotatedShape-180, RotatedShape-225, RotatedShape-270, RotatedShape-315**
ReflectedShape	**ReflectedShape-XAxis, ReflectedShape-YAxis, ReflectedShape-DiagAxis, ReflectedShape-Other**
TransformedShape-Larger	**ScaledShape-Bigger, DeformedShape-Longer, DeformedShape-LongerPart, DeformedShape-AddedPart**
TransformedShape-Smaller	**ScaledShape-Smaller, DeformedShape-Shorter, DeformedShape-ShorterPart, DeformedShape-RemovedPart**
Term from *Find-Differences* (Previous section)	
AddedShape	(**AddedShapeTypeFn** *Strict-Shape-Class*)

Table 4.2 Generalized forms for predicates in difference representations.

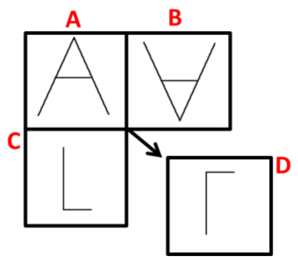
Table 4.2 gives the full list of generalized forms for terms in my representations. As the table shows, there are generalized forms for spatial relations (such as relative position), but also generalized forms for shape transformations between images. For example, the model can generalize between two different rotations, or between two transformations that make a shape bigger, such as becoming longer and having a part added on. Finally, recall that when an object is added between images, *Find-Differences* explicitly encodes its strict shape class. The shape class can be generalized to **Added-Shape**, allowing SME to map between patterns of variance with different added shape classes.

When SME performs minimal ascension, it deducts from the mapping score. Thus, a mapping that requires tiered identicality (e.g., Figure 4.10) will receive a lower score than a mapping between identical representations. If the model is considering several possible answers, it will always prefer one that produces a perfect mapping. However, if there is no perfect mapping, the tiered identicality mapping can be used. In fact, the **:differences-detected?** result will be *false* for cases like Figure 4.10. That is, the model believes the mapping to be entirely correct; it simply required some generalizing.

Infer-Shape

This operation applies a transformation to a shape to produce a novel shape. It can handle either rigid shape transformations (e.g., rotations) or deformations. However, deformations are considerably more complicated. Whereas a shape transformation involves a single quantitative transformation (e.g., a rotation with scaling), deformation involves a separate quantitative transformation for each edge. Thus, before the transformations can be applied, the operation uses SME to find the corresponding edges.

The *Infer-Shape* operation can be seen as a geometric analogy problem ("A is to B as C is to…?" or A : B :: C : ?), wherein we have a transformation between A and B, and we want to apply it to C to infer D. For example, in Figure 4.11A we have a reflection over the x-axis between shapes A and B. Applying this to shape C produces D. Suppose instead we have Figure 4.11B, where this is no transformation between A and B. Normally, if there is no transformation, the operation cannot complete. However, in this case the operation exploits a feature of analogy problems: A : B :: C : D is equivalent to A : C :: B : D (Grudin, 1980). If there is no valid transformation between A and B, the operation checks for a transformation between A and C. If possible, it applies this transformation to B to produce D.

A)


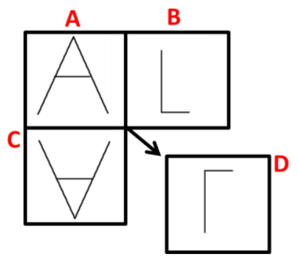
B)


Figure 4.11 Examples of *Infer-Shape* with reflections.

Figure 4.12A shows a deformation in which a part is added between A and B. With deformations, there is a separate quantitative transformation for each edge. For example, the leftmost edge in shape A grows twice as long, becoming an edge of the newly added part. To infer a deformation, *Infer-Shapes* first compares shape A to shape C. Here, it finds a reflection over the x-axis between them. It then applies the appropriate quantitative transformation to each edge; here, again, the leftmost edge grows to twice as long. It adds the new part after applying any appropriate A->C transformations; in this case, it reflects the part over the x-axis before adding it. The resulting shape is D, which has a part added at the top, instead of a part added at the bottom.

Sometimes, there is no valid mapping between the edges of A and C. For example, in Figure 4.12B, there is no mapping between the rectangle (A), and the double-headed arrow (C). In this case, the operation must make its best guess at the appropriate edges for the deformation. For example, here the A->B deformation is a lengthening of the horizontal edges. *Infer-Shapes* checks shape C and determines that it also has a pair of horizontal edges. Thus, it chooses to lengthen those edges to produce D. Similarly, in Figure 4.12C, the A->B deformation is a part lengthening going to the left. *Infer-Shapes* finds a part in shape C that can also be lengthened to the left.

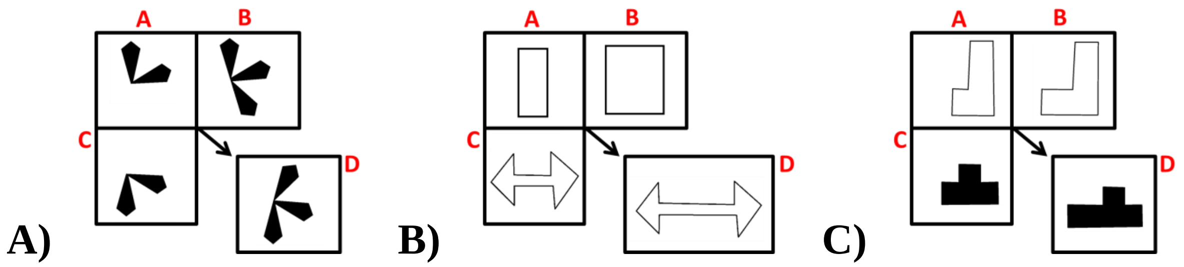


Figure 4.12 Examples of *Infer-Shape* with deformations.

Infer-Shapes also makes changes to an object's color or texture. Suppose that in the A->B comparison, the fill color is changed, or a texture gets added or removed. The operation will similarly change the fill or texture of C to produce D, when possible.

Textures are transformed according to shape transformations. If a shape is rotated 90˚ from C to D, its texture orientation will also be rotated (Figure 4.13A). If a texture is to be added in D, any transformation from B to D is applied to the texture orientation (Figure 4.13B). Thus, if a texture is being added, *Infer-Shapes* will also compare shapes B and D to check for transformations.

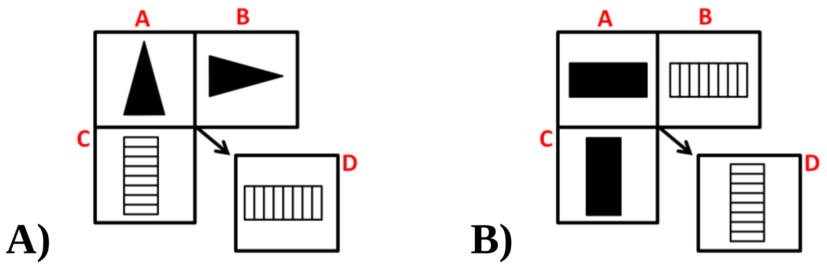


Figure 4.13 Examples of *Infer-Shapes* with texture transformations.

Input

Infer-Shape takes two input values: a shape transformation (produced by *Compare-Shapes*) and an object. It applies the transformation (A->B) to the object (C) to produce a novel object (D).

Output

Infer-Shape produces a data structure with only one item in it: the newly created object. There is no score, and there are no results. If *Infer-Shape* is unable to apply the transformation, or if there is no A->B transformation, then it returns nothing.

Infer-Image

Infer-Image applies a pattern of variance to an image representation to produce a novel image representation. Importantly, it does *not* produce a novel image. It produces a qualitative image representation. Thanks to *Infer-Shape*, this representation will include a list of object or groups, complete with their quantitative information, much like the representations produced by *Perceive*. However, there will not be quantitative information about each object's location in the image.

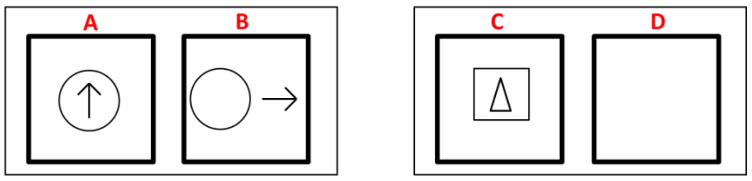


Figure 4.14 A geometric analogy problem.

Consider a geometric analogy problem (Figure 4.14). The input to *Infer-Images* would be 1) a pattern-of-variance between A and B, and 2) C's image representation. The operation would retrieve image A's representation, the first argument to the pattern of variance. Then, the operation would proceed as follows: 1) Compare A to C, and get the corresponding objects. 2) Extract the following from the pattern of variance:

a) Expression additions.

b) Expression removals.

c) Expression reversals.

d) Object transformations.

e) Object additions.

f) Object removals.

Expression additions and removals come from **changeBetweenImages** facts in the pattern of variance, while expressions reversals come from **reversalBetweenImages**. Here, two expressions are removed: (**contains** Circle Arrow) and (**centeredOn** Arrow Circle); and one expression is added (**rightOf** Arrow Circle). These changes are made to Image C's representation, substituting in the corresponding objects, to produce the new D representation.

The pattern of variance has a shape transformation (from *Compare-Shapes*) between every pair of objects—this comes from analytic image comparison. It applies these transformations to the objects in C, using *Infer-Shape*, to produce new objects for D. In this case, the square remains the same but the triangle rotations 90° clockwise. The operation then recomputes all object attributes (describing shape, fill color, etc) from the new objects and includes these in the new D representation.

Sometimes objects are added or removed in the pattern of variance. Removing an object from C is straightforward, but when an object is *added* to C, there is a question of what this object's shape should be. In this case, *Infer-Image* checks whether the added object's shape relates to one of the preexisting objects. For example, in Figure 4.15 an object is added between A and B. This object is a small circle, a scaled-down version of the pre-existing circles. Thus, when the operation adds an object to C, it will apply the scaling to a square to produce a smaller square.

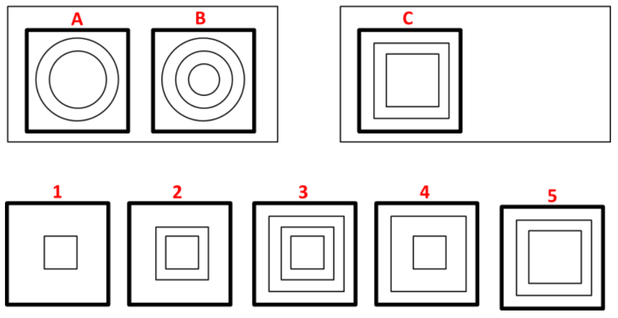


Figure 4.15 A geometric analogy problem with an added object.

Recall that patterns of variance ignore relative location attributes, i.e., concrete attributes describing where in an image an object is located (e.g., **OnTop-Element**). Because *Infer-Image* doesn't generate exact object locations, it is unable to recompute these attributes. Thus, these attributes are absent from the resulting representation.

Infer-Image becomes more complicated for 3x3 Raven's Matrices. Consider Figure 4.16A. Here, the input will be: 1) a generalization over the patterns of variance for the top two rows; 2) a pattern of variance for the first two images in the bottom row. Again, the objective is to solve for the last image. *Infer-Image* begins by choosing one of the top two rows in the generalization. It cannot use the generalization as a whole because it requires quantitative transformations between objects—these transformations are not part of the generalization. In selecting a row, it chooses the row whose objects-per-image is closest to the bottom row; in this case, all three rows have two objects per image, so either of the top rows will work.

Suppose the top row is chosen. Next, *Infer-Image* must determine the corresponding objects. It compares the top row's pattern of variance to the bottom row's. Though the bottom row has one less image, both rows represent differences between the first two images. The mapping between these differences should be sufficient for finding corresponding objects. Here, the circle maps to the trapezoid and the arrow maps to the rectangle because in each row, the smaller object changes from being left of the larger object to being contained inside it.

Once corresponding objects are found, the operation applies the top row's Image-2->Image-3 differences to Image-2 in the bottom row to compute that row's Image-3, as described above. Here, the rectangle rotates 90˚ and changes from being inside the trapezoid to being right of the trapezoid.

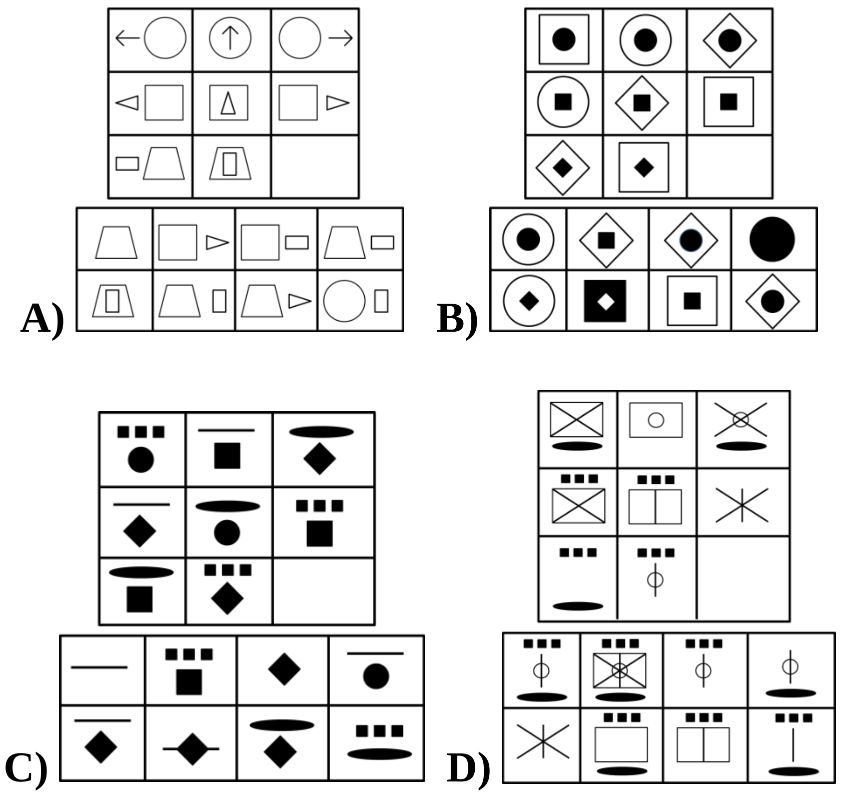


Figure 4.16 Four 3x3 Raven's Matrix problems.

Recall that there are different types of patterns of variance. While Figure 4.16A involves the most basic kind, a pattern of variance may be literal, and it may be simple. *Infer-Image* applies a basic principle for these types of patterns: compare the complete row (e.g., the top row) to the incomplete row (the bottom row). Anything in the complete row that isn't found in the incomplete row belongs in the final image. For example, in Figure 4.16B, the diamond and square images in the top row align with the diamond and square images in the bottom row. The circle image is missing, so the model infers a circle image for the bottom row.

Figure 4.16C involves a simple, literal pattern of variance. Recall that in a simple pattern of variance, the images are broken up into their objects. Thus, *Infer-Images* looks for individual objects in the top row that are missing from the bottom row. Here, the line and the circle are missing, so the model infers a new image with a line and a circle.

Figure 4.16D involves a simple, differences pattern of variance. Recall that here, the images are broken up into their objects, but the pattern still describes differences between objects. The top row's pattern of variance describes a set of objects, each of which remains constant between two images but is removed in the third image (for example, the rectangle is in only the first two images, whereas the circle is in only the last two images). Again, *Infer-Images* compares the top and bottom rows and asks what is missing in the bottom row. Here, it asks what object *differences* are missing. The group of squares is constant across the two rows; therefore, the missing difference is the object removal—it should be removed in the last image. The ellipse is in the first row but removed in the second; therefore, the missing difference is the object remaining constant—it should be present in the last image. Based on filling in the missing differences, *Infer-Image* determines that the final image should contain an ellipse, a circle, and a vertical line.

Infer-Image requires one additional step for normal literal patterns of variance. In Figure 4.16B, the bottom row is missing the circle image. However, the model cannot simply insert the top row's circle with a circle inside it. Here, the top-row images all have black circles, while the bottom-row images all have black diamonds. Thus, *Infer-Image* must call itself recursively. First, it compares the top-row image with a square to the bottom row image with a square, using *Find-Differences* to compute a pattern of variance. This pattern of variance describes a change in which the black circle becomes a black diamond. Then it applies this pattern to the missing image, the image with the large circle, using *Infer-Image*. This produces the necessary image, a large circle around a black diamond.

Importantly, simple patterns of variance don't describe relations between objects. Thus, when *Infer-Images* operates on simple patterns, it produce an image representation that lacks spatial relations—it specifies only what objects are present in the image.

Fail Conditions

Image inference can provide an elegant solution to geometric analogy and Raven's Progressive Matrix problems. However, sometimes there is not enough information to infer the missing image. It is important to recognize when this is the case. Thus, *Infer-Images* performs several checks to determine if it has failed. In the following descriptions, I again use the basic analogy example ("A is to B as C is to…?").

Infer-Images focuses on adding expressions to C to produce D', the answer image representation. It returns a failure whenever it is unable to add an expression. It might fail because an expression to be added describes an element not found in C. It might also fail because an expression to be reversed is not found in C.

Infer-Images must also produce the correct set of objects for D'. It returns a failure when it is unable to do so. It might fail to remove an element because C lacks that element, or it might fail to transform a C object into a D object because there is no valid A->B transformation, or because a deformation cannot be applied.

After transforming a C object into a D object, *Infer-Images* compares the objects to determine whether the appropriate transformation was achieved. This check leads to a controversial prediction that appears to match the human data. Consider Figure 4.17. Given this problem, *Infer-Images* will apply a rotation to the octagon and then check whether the rotation had the desired effect. It will find that it did not—because of the octagon's symmetry, the rotation produces another identical octagon, rather than a rotated octagon. Therefore, *Infer-Images* will return a failure, and the routine will attempt a different approach.

Thus, the model explains why people have difficulty with problems like Figure 4.17: due to the octagon's symmetry, rotating it produces the same shape, rather than a satisfyingly different shape. See Chapter 6, the Geometric Analogy chapter, for a detailed discussion of problems like Figure 4.17.

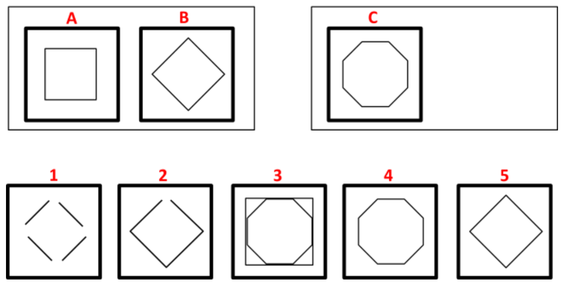


Figure 4.17 A geometric analogy problem with an unclear rotation.

Finally, sometimes a problem is too complex to even attempt *Infer-Images*. For example, consider the Raven's problem in Figure 4.18. This problem requires complex perceptual reorganization. In each row, the routine must break the objects down into edges, and then group the edges back together based on which edges are common across the images. The routine determines that in each row, there are two horizontal edges in the first image, two vertical images in the second, and neither in the third. Everything else is constant across the row, and so it gets grouped together to form a single object.

However, on the bottom row, there isn't enough information for perceptual reorganization—the third image is missing, so it isn't clear how the first two images should be reorganized. Admittedly, a person might figure this out by comparing the individual edges across rows. However, this goes the model's current level of reasoning.

More concretely, when *Infer-Images* is comparing patterns of variance that are neither literal nor simple, it checks the images in the top row that correspond to images in the bottom row (i.e., the first two images). If they each have a :dynamic focus, meaning the *Perceive* operation has been updated with segmentation or grouping commands, then the *Infer-Images* concludes that the problem is too complex.

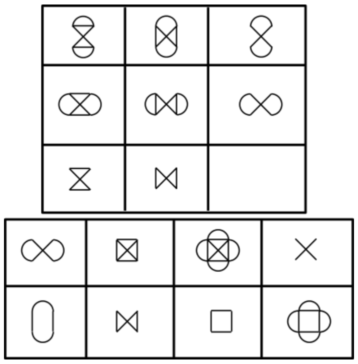


Figure 4.18 A Raven's Matrix problem requiring complex perceptual reorganization.

Input

Infer-Image normally takes a pattern of variance and a target image. Optionally, the first argument can be a generalization over patterns of variance, and the second argument can be a pattern of variance over images. Finally, there is an optional third argument: the location of the inferred image (e.g., the lower right cell in a Raven's matrix). This information is used only for displaying results to the researcher.

Output

This operation produces a data structure containing a new image representation. Again, an image representation is a qualitative representation (a list of predicate calculus expressions) and a set of objects.

Results

:apply-problems In most cases, *Infer-Image* will produce an output even if it hits a fail condition (the exception is the last fail condition described). Any failures will be listed here.

Detect-Texture

The above operations work on abstract, qualitative representations. However, sometimes these representations are not enough. Some early problems in Raven's Standard Progressive Matrices, section A, require low-level texture detection. For example, Figure 4.19A isn't about the objects and relations between objects; it is about the repeating '+' pattern. Recall that *Perceive* can detect textures, but only repeating lines of a single orientation. *Detect-Texture* can identify larger macro-textures.

Detect-Texture seeks out a repeating quantitative pattern along one strip of an image. To locate such a strip, it uses *Locate* with inputs such as ("Problem" :y 1 :referent "Answer" :x 2 :y 2). Recall that "Problem" is the name of the large rectangle, while "Answer" is the name of the missing portion. These *Locate* arguments point to the top row (:y 1), where "Answer" is in the lower right position (:x 2 :y 2). Because no :x value is specified for the location, *Locate* returns a corridor stretching the entire length of the image (see Figure 4.19B, upper gray rectangle). This location will be used to detect a texture. Similarly, *Locate* can find the lower corridor containing the missing portion (Figure 4.19B, lower gray rectangle). This will be used to evaluate each possible answer.

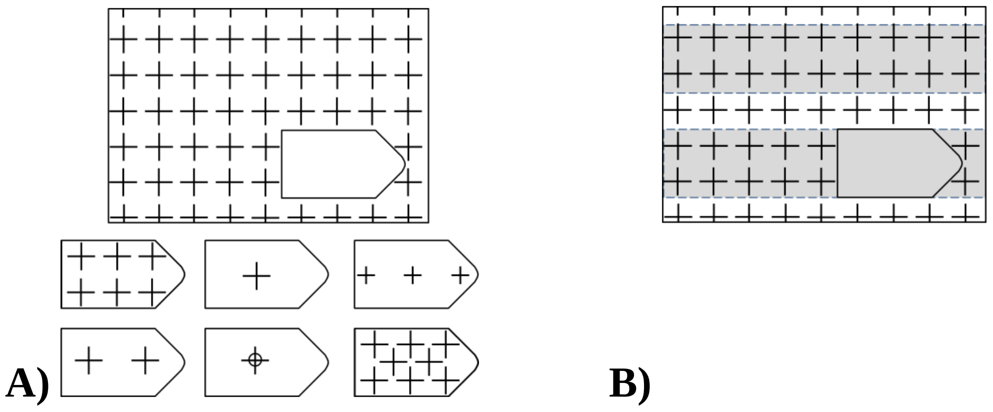


Figure 4.19 A Raven's Matrix problem requiring texture detection.

Given a corridor along an image, *Detect-Texture* prints this corridor onto a bitmap. That is, it creates an image with a value of 0 or 1 at each point, depending on whether ink was found at that point. Thus, it loses all qualitative information and all object information, producing a purely quantitative representation of the corridor. This might be similar to an early, low-level visual representation in the human brain.

Detect-Texture uses autocorrelation (Oppenheim & Schafer, 2009), a common approach from signal processing, to detect repeating patterns in a corridor. The idea is to divide the signal into windows and check whether consecutive windows correlate with each other. One varies the window length to find the best correlation. Here, each window is a portion of the corridor's total length.

Detect-Texture computes two signals from a window: rows and columns. For rows, it takes each row of the window and adds up the pixels that are "on." Thus, the value will be highest for rows going through every plus's horizontal line, and it will be zero for rows between pluses. Similarly, it takes each column of the window and adds up the pixels that are "on." The value will be highest for the columns going through two pluses' vertical lines, and it will be zero for columns between pluses. In computing the correlation between two windows, *Detect-Texture* takes the lower of the correlation between rows and the correlation between columns.

Detect-Texture takes four key parameters: a minimum window length, a maximum window length, a window variability, and a minimum autocorrelation score. It moves along the corridor, finding optimal windows. For the first two windows, it finds the length (subject to the minimum and maximum) that maximizes the correlation score. This is the **:window-size**. For the correlation with each consecutive window, it allows the window length to vary from **:window-size** according to the window variability—this should be a small value. If every consecutive pair achieves the minimum autocorrelation score, it returns the **:window-size**.

To evaluate a possible answer, one passes *Detect-Texture* the lower corridor and specifies several directives: **:target-size** indicates a **:window-size**, computed from the upper corridor. **:insertion-reference** is the location where something is to be inserted, e.g., the "Answer" object. **:insertion-target** is the location of the answer being inserted. Given these directives, *Detect-Texture* will insert a possible answer into the missing space, set the **:window-size**, and then move across the corridor, checking whether consecutive windows achieve the minimum autocorrelation score.

Input

The input is the location of the corridor. As with *Perceive*, one can either pass the location itself or pass a list of *Locate* keywords.

Parameters

The default values for the parameters below have all been empirically determined, based on what works best for Raven's Matrix problems. Window lengths are given as a percentage of the overall corridor length.

:min-auto-corr-window/:max-auto-corr-window The minimum/maximum length for windows along the corridor. The default values are .05/.2.

:auto-corr-window-var The variability allowed in window length. The default value is .01.

:min-auto-corr The minimum autocorrelation between windows. Correlation values can vary from 1 (perfect correlation) to 0 (signals are entirely unrelated) to -1 (perfect reverse correlation). The default minimum is .85.

Directives

:target-size Specifies a window length, rather than having the operation compute this from the first two windows.

:insertion-target Specifies the location of an image that is to be inserted into the corridor.

:insertion-reference Specifies where in the corridor the image is to be inserted.

Output

This operation only returns if all window correlations are above **:min-auto-corr**. The only information in the output is the result below.

Results

:window-size The window length that produced the optimal autocorrelation. Can serve as the **:target-size** directive in a future *Detect-Texture* operation.

4.3 Other Operations

The operations above do most of the work during problem-solving. However, there are several smaller operations that can support the spatial routine.

Survey

Given a location, this operation surveys all the objects in that location, gathering the distribution of sizes and line thicknesses. This information is used when *Perceive* computes relative size attributes.

Input

This operation takes a location, or a list of *Locate* keywords. If no location is given, it will survey all objects across the entire sketch.

Parameters

:sizes? Survey the distribution of sizes?

:line-widths? Survey the distribution of line thicknesses?

Output

This operation produces no output. Rather, the distribution information is saved in the *routine working memory* (see below), where it can be accessed by future *Perceive* operations.

Recognize-Shape

This operation registers an object in the shape registry. It is rarely called directly in a spatial routine. Rather, other operations that require shape class information (e.g., *Compare-Images*) will call it.

Inspect

This operation inspects a sketch-lattice to get information about its dimensions.

Input

The operation takes the location of a sketch lattice.

Results

:num-rows The number of rows in the sketch-lattice.

:num-cols The number of columns in the sketch-lattice.

Access

This bookkeeping operation simply returns whatever is passed in as its input. It is a useful way to store the input (which might be a list of things) in the routine working memory (see below).

Display Operations: Select and Abandon

There are two operations that have no effect on what a spatial routine does. Rather, they affect what is displayed to the researcher. Thus, they are useful only for debugging and demonstration purposes.

Firstly, *Select* draws a square around an image's location. It is useful for displaying which answer the spatial routine has chosen.

Secondly, *Abandon* indicates that the routine has given up on a particular operation's result (most likely an inferred image representation). It tells the system to not bother displaying this operation's result unless it's specifically requested.

4.4 Spatial Routine Language

A spatial routine is a list of operations. These operations are executed in sequence. As described above, many operations produce a data structure containing the results of that operation. This data structure can be stored in the *routine working memory* (RWM) and accessed at a later time. The RWM also stores the shape registry, the texture registry, and the results of the *Survey* operation.

I now provide a primer in the spatial routine coding language. This coding language is built into the LISP coding language, and as such, it adopts two LISP conventions: 1) All code and data is written as lists surrounded by parentheses. 2) Whenever a Boolean value is checked, nil indicates *false*. t or any other value indicates *true.*

Table 4.3 shows the template for spatial operation calls, along with a couple examples. The first item is the *pre-check*, a spatial expression that must return non-nil before the operation is called (see "spatial expressions" below). If the pre-check fails, the routine skips this operation and continues to the next. The next item is a keyword giving the operation type. Next there are three lists for the inputs, parameters, and directives. Finally, there may be optional arguments.

Template	Image Encoding	Image Update
(*pre-check*	(t	((:eq (:result :num-elements :my-image) 1)
operation-type	:perceive	:perceive
inputs	(:answers :x 1 :y 1)	(:update :my-image)
parameters	(:focus :objects :group-edges? t :textures? t)	(:focus :edges)
directives	Nil	nil
***optional extra arguments*)**	:store-var :my-image)	:store-var :my-image)

Table 4.3 A template and examples of operation calls.

In the Image Encoding example, the pre-check is t. This means the operation will always be called. The operation type is **:perceive**, meaning the *Perceive* operation. The inputs are a list of *Locate* keywords, indicating the leftmost, topmost image in the :answers location (presumably some previously stored location). The parameters indicate that that this will be encoding at the object-level. Two additional parameter values are specified. All other parameters will take their default values. Finally, one optional extra argument is specified. **:store-var** indicates that this operation's result should be stored in RWM under the specified name, in this case :my-image.

Now consider the Image Update example. Here, the pre-check is more complicated. **:eq** indicates we are checking whether two values are equal (see below). (**:result :num-elements** :my-image) retrieves the object stored as :my-image and gets its **:num-elements** result. So, this pre-check says to only proceed if :my-image contains only a single element. Here, the input is (**:update** :my-image). This means we will retrieve the image representation stored as :my-image and update it. Under parameters, we see that we are changing to an edge-level representation. All other parameters will remain the same. Finally, this operation is to be stored under the same name. Thus, it will overwrite the previous operation in the RWM.

Suppose these two operation calls were listed in sequence in a spatial routine. The first call would generate an image representation at the object level and store it in RWM. The second indicates that if there is only a single object in the image, we should change to an edge-level representation.

I now describe the coding language in greater detail. First, I discuss spatial expressions, expressions that SRS must resolve to determine, for example, whether an operation's pre-check has passed. Next, I list the optional extra arguments for operation calls. After this, I explain the control structures that support looping and backtracking. Finally, I bring these together in an example routine.

Spatial Expressions

Spatial expressions are symbols or lists written by the user in the spatial routine. During execution, SRS resolves them to compute some value. Every precondition, every input list, and even every parameter or directive value is a spatial expression. Below, I list many of the common spatial expressions and explain how they are resolved. Note that several spatial expressions take an optional argument, a symbol referring to some previous operation stored in the RWM. In such cases, if one doesn't include this argument, SRS uses the last completed operation. For example,

(**:result :num-elements**)

will return the **:num-elements** result from the last operation completed. For simplicity, I include a * at the end of *Symbol* to indicate such an optional argument.

t This always means *true*.

nil This always means *false*.

:null This can also mean *false*.

Symbol Look this symbol up in the RWM. If there is an operation stored under this name, return its output. Otherwise, just return the symbol.

(**:stored?** *Symbol*) Returns **t** if something is stored in RWM under this symbol name.

(**:single-value?** *SymbolOrExpr**) Returns **t** if *SymbolOrExpr* resolves to something that is non-nil and not a list.

(**:stored-under** *Symbol1 Symbol2*) The RWM supports an optional two-level hierarchy for storing operation output (see below). This returns the operation stored as *Symbol2* under the category of *Symbol1*.

(**:result** *Type Symbol**) Returns result-type *Type* from the operation stored under *Symbol*.

(**:parameter** *Type Symbol**) Returns parameter-type *Type* from the operation.

(**:directive** *Type Symbol**) Returns directive-type *Type* from the operation.

(**:score** *Symbol**) Returns the score from the operation.

(**:input** *n Symbol**) Returns input number *n* from the operation.

The following expressions implement some basic Boolean and mathematical operations.

(**:not** *Expression*) Resolves *Expression* and returns **t** if *Expression* is **nil**, or **nil** if it is non-nil.

(**:and** *Exp1 Exp2 … ExpN*) Resolves each subexpression. If all of them are non-nil, returns the last resolved expression. Otherwise, returns **nil**.

(**:or** *Exp1 Exp2 … ExpN*) Resolves each subexpression until one of them is non-nil. Returns that one.

(**:eq** *Exp1 Exp2*) Performs numerical comparison. Returns **t** if the two subexpressions resolve to equal values (often numbers).

(**:gt** *Exp1 Exp2*) Returns **t** if *Exp1* is greater than *Exp2*.

(**:lt** *Exp1 Exp2)* Returns **t** if *Exp1* is less than *Exp2*.

The remaining expressions implement some more advance list operations.

(**:every** *Expression Symbol*) *Symbol* should resolve to a list of operation outputs. This will iterate over those outputs and resolve *Expression* while treating each of them as the last operation. Returns **t** if *Expression* resolves to non-nil every time. An example might be:

(**:every** (**:eq** (**:parameter :focus**) **:edges**) :top-row-reps)

This will return **t** if every image representation in :top-row-reps has an edge-level focus.

(**:some** *Expression Symbol*) As above, but this only requires that *Expression* resolve to non-nil for one of the operation outputs. Returns the first non-nil resolution of *Expression*.

(**:notany** *Expression Symbol*) As above, but this returns **t** if *Expression* resolves to *nil* for all the operation outputs.

(**:remove-from** *Expr1 Expr2*) *Expr2* should resolve to a list, while *Expr1* should resolve to a single item. This returns the list with that single item removed from it.

(**:string** *Expr1 Expr2 … ExprN*) Returns a string of text containing the resolutions of the subexpressions. Mostly useful for demonstration and debugging purposes.

If none of the above expression types are triggered, then an expression remains unchanged. For example, a number resolves to itself.

Optional Extra Arguments for Operations

Operation calls can take several optional arguments. Some affect the spatial routine's behavior, while others are only for demonstration and debugging purposes.

:post-check *Expression* Only return this operation's output if the *Expression* resolves to non-nil. For example

(**:gt** (**:result :num-elements**) 1)

might indicate that a *Perceive* operation must produce an image representation with more than one element.

:min-score *Number* This specialized post-check specifies a minimum score for the operation.

:store-var *Name* Store this operation's output in the RWM under *Name*.

:store-under *Name2* Normally, operation output is stored as a single name, specified by **:store-var**. However, for complex routines, a two-level hierarchy of naming can be useful. This argument, combined with the last, indicates that the output should be stored as *Name* under the category of *Name2*.

The remaining optional arguments are for demonstration and debugging.

:name "Name" Indicates a name (usually a text string) for the user display.

:comments "Description" Indicates comments on the operation for the user display.

:display? t/nil Display the results of the operation?

:reset-and-display? t/nil Reset the display and then show the results of the operation?

:animate? t/nil Animate the operation (shape comparison or visual inference)?

Control Structures

The SRS coding language includes several control structures that support iteration, backtracking, and modularity. Among other things, they can model selecting from a list of possible answers, i.e., making a *forced choice*. This is generally done by assigning a score to each answer and picking the one with the highest score. However, when several answers receive very similar scores, it is not clear that a person could distinguish between them. In such cases, the model returns a list of the indistinguishable choices, rather than guessing. This behavior can be modified by changing the **gt-threshold**, the threshold at which one score is noticeably greater than another (see section 5.3).

The syntax for these control structures is similar to the syntax for operation calls (see Table 4.4). A control keyword indicates that this will be a control structure rather than an operation call. In the place of input, parameters, and directives, there is a return keyword, indicating what the control structure's output will be. Again, there are optional extra arguments—and some control structures have their own specialized optional arguments. Finally, there is the body for the control structure: a list of operation calls. I describe each of these below.

Template	Iterate Statement	Or Statement
(*pre-check*	(t	(t
control keyword	(:iterate-over :answers :answer)	:or
return keyword	:best-score	(:min-score .80)
optional extra arguments	:store-input :best-answer	:store-var :good-comparison
list of operation calls)	((*Operation-Call_1*) (*Operation-Call_2*) ….)	((*Operation-Call_1*) (*Operation-Call_2*) ….)

Table 4.4 A template and examples of control structures.

Control keywords

There are four types of control structures, selected by their control keyword.

:and An And Statement is simply a list of operations that are executed in sequence. Typically, a spatial routine is an And Statement at the top level.

:or An Or Statement is similar to an And Statement, except the operations are treated as though they were executed in parallel. This means that, for purposes of resolving spatial expressions, the *last operation* is always the operation that completed before the Or Statement began. Whereas in an And Statement, each operation builds on the work of the previous operation, here the operations are testing out different alternatives.

:repeat A Repeat Statement executes the operations in the body in sequence and then checks whether a condition (specified by the return-keyword) is met. If the condition is not met, it repeats the body.

(**:iterate-over** *Symbol SymbolOrExpression*) An Iterate Statement iterates over a list, specified by *SymbolOrExpression*. For each value in the list, it temporarily stores that value under the name *Symbol* in the RWM and then executes the operations in the body in sequence.

Return keywords

For Repeat Statements, the return keyword is a post-condition or min-score saying when SRS can stop repeating. Once the post-condition is met, the statement returns the output from the last operation.

For the other three control structures, there can be either a single keyword or a list of keywords from the set below. These keywords indicate when the control structure is finished and which operation's output should be returned. Note that for And and Or Statements, the control structure is selecting the output from one of the operations in the body. For Iterate Statements, it is selecting the output from one of the iterations, where all the operations in the body are called for each iteration.

:last Return the output from the last successfully completed operation.

:contingent Similar to :last, but this requires that every operation in the sequence is successful. If any of them fails, the control structure returns **nil**. Currently defined only for And Statements.

:best-score Return the output with the best score.

:worst-score Return the output with the worst score.

(**:first-score** *Number*) Return the first operation output to achieve a score of at least *Number*, or else the highest-scoring output.

(**:first** *Expression*) Return the first operation for which *Expression* resolves to non-nil, or else nothing. Note that this could be combined with **:last** :

(**:first** *Expression* **:last**)

This would indicate that it should return the last operation output, if none achieves *Expression*.

The remaining keywords can be used together with **:best-score** or **:worst-score**.

:forced-choice Indicates that we're making a forced choice in choosing among the possible outputs. If one output is noticeably better than all others (its score is more than **gt-threshold** beyond all others), then we'll return it. Otherwise, we'll return a list of the best outputs, to indicate that we couldn't choose between them. Without this keyword, the control structure will always pick the best score, even if it's highly similar to the second-best.

:!prefer-single When the control structure is choosing the best or worst score out of several options, it is possible that some of the options fail to return a single output, instead returning a list of outputs.

Normally, a single output is always preferred, regardless of score. However, this keyword indicates that we should not prefer single values. Thus, if one iteration returns a list of three outputs and another iteration returns a single output with the same score, and these are the highest scores, then we will return a combined list of four outputs.

Optional Extra Arguments

There are a few specialized optional arguments for control structures.

:store-input *Name* This applies only to Iterate Statements. Recall that these iterate over a list of values. After selecting the best iteration, Iterate will store the value that produced that iteration in RWM under *Name*.

:store-second-var *Name* This applies if we're selecting a best-scoring operation with **:best-score** or **:worst-score**. The output of the second best-scoring operation will be stored under *Name*. This is primarily used for demonstration and debugging.

:store-second-input *Name* Again, this applies to Iterate Statements. The value that produced the second-best output will be stored under *Name*. This is primarily used for demonstration and debugging.

:iteration-comment "Description" Again, this applies to Iterate Statements. In the user display, each iteration will receive this comment. This is used only for demonstration and debugging.

Example Routine: Bongard Problems

Table 4.5 shows an example spatial routine. This routine solves Bongard problems (Bongard, 1970), in which one is shown two sets of images and asked how one set differs from the others. For example, in Figure 4.20, the left objects have a white fill, while the right objects have a black fill.

The spatial routine implements a basic strategy: 1) *Generalize* over the first array, to determine what is common across them. 2) *Generalize* over the second array to determine what is common across them. 3) *Find-Differences* between the two generalizations to determine how they differ. Unlike the routines described in future chapters, this routine incorporates no strategic backtracking. Thus, it can solve only the simplest Bongard problems, such as Figure 4.20.

Note that the overall routine is written in an And Statement. This is a common approach. However, because there is no backtracking, no other control structures are needed. In addition, no updates are needed, so the directives are always empty, i.e., **nil**.

Let us consider the routine in greater detail. The And Statement around the entire routine has a return keyword of **:last**, so it will return the last successful operation. In the body of the And Statement, the first operation called is *Survey*. This operation has no input (**nil**), so it will survey the entire sketch, gathering the distributions of sizes and line widths.

After this, there are two *Locate* operations. Their inputs indicate that they're locating the leftmost and second leftmost (in this case, rightmost) sketch lattices. The lattices' locations are stored in the RWM as :left-lattice and :right-lattice.

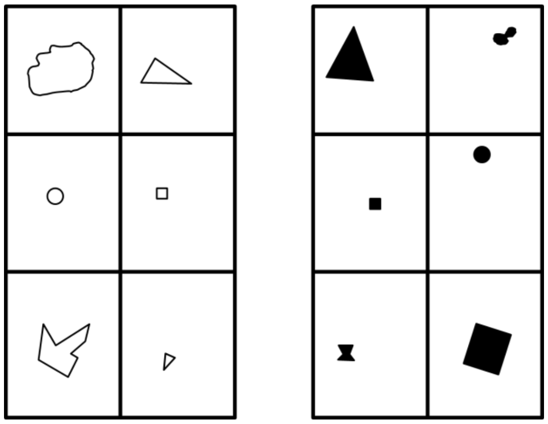


Figure 4.20 A Bongard problem from (Bongard, 1970).

```
(t :and :last
      :reset-and-display? t
      ((t :survey
           nil (:sizes? t :line-widths? t) nil)

       (t :locate
          (:x 1) nil nil
          :store-var :left-lattice)

       (t :locate
          (:x 2) nil nil
          :store-var :right-lattice)

       (t :perceive
          (:left-lattice)
          (:focus :groups :group-edges? t :shape-features
                (:fill-colors :border-colors :border-width
                :relative-size :symmetry :other-attributes))
          nil
          :name "Left "
          :store-var :left
          :comment "Encode left images")

       (t :perceive
          (:right-lattice)
          (:focus :groups :group-edges? t :shape-features
                (:fill-colors :border-colors :border-width
                :relative-size :symmetry :other-attributes))
          nil
          :name "Right "
          :store-var :right
          :comment "Encode right images")

       (t :generalize
          (:left) nil nil
          :name "Left Generalization"
          :store-var :left-gen
          :comment "Generalize over left images.")

       (t :generalize
          (:right) nil nil
          :name "Right Generalization"
          :store-var :right-gen
          :comment "Generalize over right images.")

       ((:and (:stored? :left-gen) (:stored? :right-gen))
        :find-differences
        (:left-gen :right-gen)
        nil nil
        :name "Differences"
        :store-var :answer
        :comment "Find differences.")))
```

Table 4.5 A spatial routine for solving Bongard problems.

Next, *Perceive* is called on each lattice's location. When *Perceive* is called on an entire lattice, it runs on each image in the lattice, returning a list of image representations. Note that *Perceive* takes several parameters. The **:focus** indicates that we're encoding at the **:groups** level. The **:shape-features** parameter allows us to list several types of shape features that should be encoded. Here, we'll be encoding the colors and sizes of the borders and interiors, as well as symmetry. **:other-attributes**, refers to a large set of shape features from Chapter 3 (Table 3.3).

After this, *Generalize* is called on each list of image representations. Note that *Generalize* takes no parameters or directives here; its default behavior is sufficient. The *Generalize* operations produce two generalizations, stored in RWM as :left-gen and :right-gen. Finally, these generalizations are compared by *Find-Differences*, which also takes no parameters or directives. Run, on Figure 4.20, *Find-Differences* produces a pattern of variance (Table 4.6)[5].

```
((ElementInARowFn 0) Image-1)
((ElementInARowFn 1) Image-2)

(changeBetweenImages
  (isa (ElementInImageFn Element-3 Image-2)
       (ObjectsColoredFn BlackColor))
       Image-1 Image-2)

(changeBetweenImages
  (isa (ElementInImageFn Element-3 Image-1)
       (ObjectsColoredFn WhiteColor))
       Image-2 Image-1)
```

Table 4.6 Pattern of variance for Figure 4.20.

[5] Here I give the full syntax for the pattern of variance, rather than the simplified syntax used in Table 4.1 (see footnote 4).

According to this Pattern of Variance, the difference in one direction is that the objects have a black fill, while the difference in the other direction is that the objects have a white fill.

4.5 Gathering Data

Spatial routines can be executed on psychological stimuli and compared against human performance. However, SRS does more than evaluate whether a strategy can solve a given problem. Its data-gathering mechanisms help demonstrate *how* a strategy solves a given problem.

These mechanisms are also useful for evaluating working memory load. There are no working memory constraints in SRS. However, SRS can provide objective measures of working memory load, such as the number of elements in each image representation or in each pattern of variance. These measures can be compared against human performance, as we shall see in later chapters.

Operation Trace

The operation trace stores all the operation calls made during a problem-solving episode. It also stores operation retrievals. There are two cases where an operation's output will be retrieved from the RWM: 1) the routine might reference the name under which the operation was stored; 2) the routine might call an operation with identical inputs, parameters, and directives to a previous operation call. In either case, the operation output will be retrieved, rather than recomputed. The operation trace records the number of times each operation's output is retrieved.

While the operation trace can be inspected directly, it also drives the Routine Inspector. The Routine Inspector is a graphical user interface which lets the user step through each operation and view its results. Figure 4.21 shows the Routine Inspector describing the operations involving Image B, the second image from the left, in geometric analogy. Because the *Compare-Images* operation is selected, the display shows the corresponding elements in the two images.

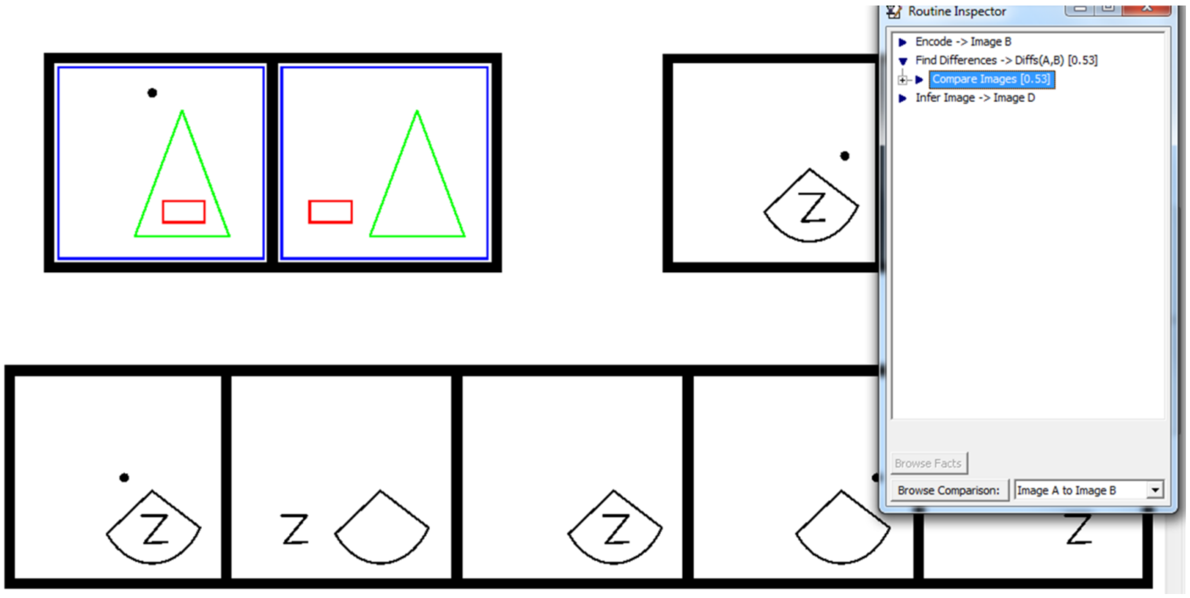


Figure 4.21 The Routine Inspector on a geometric analogy problem.

Routine Working Memory

After a problem-solving episode, a researcher can inspect the RWM directly to see the output of each stored operation. This is useful for checking working memory load: one can inspect the number of elements or expressions in the image representations, the generalizations, or the patterns of variance. It is also useful for viewing how the routine solved a problem: the parameters and directives indicate which particular strategy produced the solution.

Ablation

Suppose we want to know which problems require a particular cognitive operation, such as computing shape transformations. One way to measure this is via ablation: removing the routine's ability to perform that operation, and checking which problems it can no longer solve.

SRS incorporates several optional keywords to perform ablation on a spatial routine. These keywords are utilized when the routine is executed on a problem. They are:

:required-parameters

Sets required parameter values for particular operations. These values overwrite any parameter values found in the routine. For example, one could require that the **:shape-trans-relations?** parameter be **nil** in *Perceive*. This means that *Perceive* will never compute shape transformation relations.

:invalid-parameters

Specifies parameter values for particular operations that will always fail. For example, if **(:focus :edges**) is an invalid parameter for *Perceive*, then any *Perceive* operation that attempts to build an edge-level representation will fail, returning **nil**.

:invalid-directives

Specifies directives for particular operations that will always fail. For example, if **:segment-edges** is an invalid directive for *Perceive*, then an object can never be broken down into its edges, so complex perceptual reorganization is impossible.

5. Simulations

I built spatial routine models for three tasks: geometric analogy, Raven's Progressive Matrices, and the oddity task (Figure 5.1). My objective was to use identical representations and processes whenever possible. This is particularly important for the representations. Each of the three tasks constrains the range of possible representations. By identifying a representation scheme that works across the three tasks, we can provide evidence that the scheme is sufficient for modeling human spatial problem-solving. Thus, we can make significant progress on determining how people represent space.

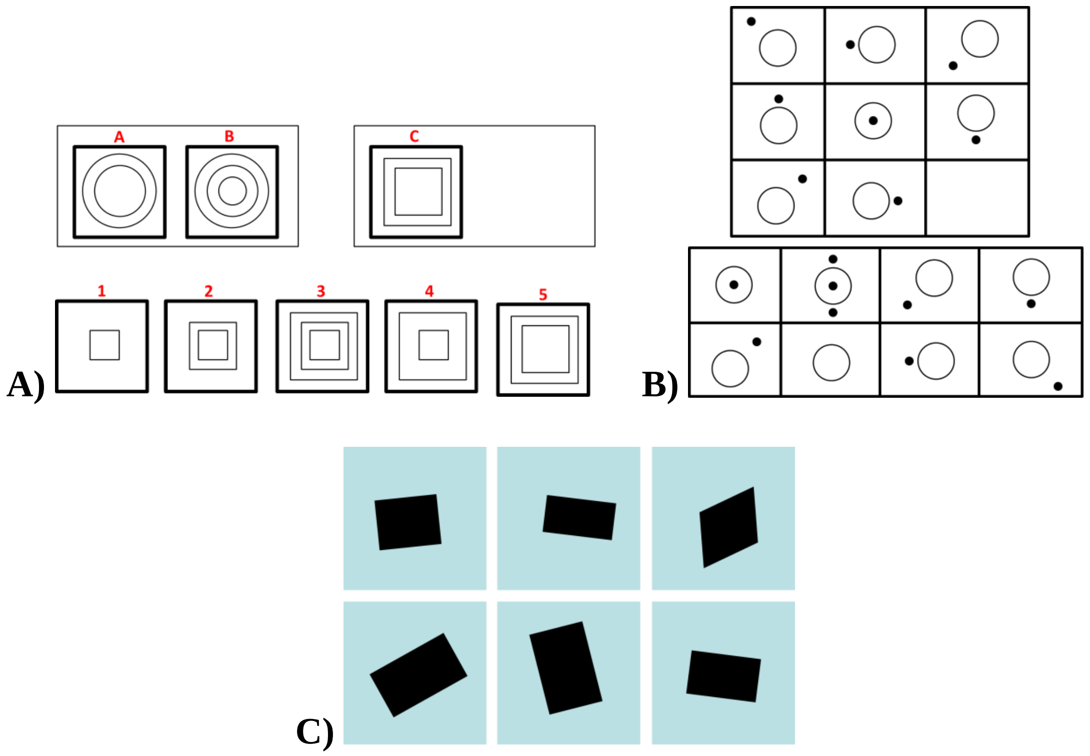


Figure 5.1 A: Geometric analogy problem (Evans, 1968). B: Raven's Matrix problem. C: Oddity task problem (Dehaene et al., 2006).

In this chapter, I discuss the commonalities across the task models. I begin with the representation scheme. I then compare the three tasks, showing how geometric analogy and Raven's Progressive Matrices are particularly similar. Next, I describe a sensitivity analysis, in which I determined the range of appropriate values for two numerical parameters used by the models. Lastly, I introduce the analyses I will be conducting on each simulation's results. In the following chapters, I cover each task model in detail.

5.1 Representation

Historically, representation has presented a major problem for cognitive modeling. As researchers, we don't know the form or content of people's mental representations. Thus, when we build a model, we must make assumptions about both the process and the representations that serve as input to that process. One common approach (Gentner, Ratterman, & Forbus, 1993; Goldstone & Medin, 1994) is to hand-code the representations. That is, given a set of stimuli (e.g., images or stories), a person constructs each stimulus' representation in a form that their process model will understand.

The problem with this approach is that the coder can tailor each representation to meet the needs of the process model, rather than using representations that are general or well-supported by the literature. As a result, the model is largely unconstrained by any representation considerations. Because the model is unconstrained, there is less we can learn from it, even when it perfectly matches human performance—it may behave like a person, but would it behave like a person given the same input?

One improvement is to automatically encode the stimuli (Hofstadter, 1995; Forbus et al., 1998; Forbus et al., 2011). That is, one builds a (possibly) separate model for the encoding process. When chained together, these two models constrain each other (Forbus, 2001): the encoding process must produce representations that the reasoning process can use.

Even this approach may not be sufficiently constrained. An encoding process could build representations that are only useful in the task being modeled. If so, it tells us little about human mental representations, which presumably are used across many tasks.

Ideally, one should show that a single encoding process produces representations that work across many tasks. This places more constraints on the encoding process, as its representation must be sufficiently general to support all the tasks. It also places more constraints on the task processes, as they must utilize representations that weren't constructed solely for them.

My goal is to show that a single visual encoding scheme supports all three spatial problem-solving tasks. I have already described the spatial vocabulary in this encoding scheme (see section 3.2). Below I briefly explain the spatial routine approach for producing that vocabulary.

Visual Encoding in SRS

Spatial Routines for Sketches (SRS) depends on the user to segment an image into a set of objects. Thus, the first step in encoding is performed by the researcher. All my stimuli were initially constructed in PowerPoint, which supports drawing shapes with perfect straight edges and curves. Most of the oddity task stimuli were constructed by the original researchers, while I reproduced the geometric analogy and Raven's Matrices stimuli (see the upcoming chapters for more information).

In constructing stimuli, I drew each closed shape as a single object in PowerPoint. In the absence of closed shapes, I drew edges as part of the same object if they were connected and they lacked ambiguous forks (i.e., places where three or more edges meet). When there were forks, I drew separate objects, respecting the Gestalt law of good continuation (Kellman, 2000). For example, when edges form an "X" shape, one sees this as two straight edges intersecting, rather than as one corner above another. In essence, this means that I manually performed *contour integration* (Field, Hayes, & Hess, 1993), a low-level visual process in which edges are joined together to form objects.

It was sometimes impossible to join the appropriate edges together as a single object in PowerPoint. In such cases, I drew them as separate objects, imported them into CogSketch, and then used CogSketch's *merge* function, which combines two objects into a single object.

The above approach means that when an object contained textures (e.g., Figure 5.2), each line of the texture was generally drawn as a separate object. After these objects were imported into CogSketch, the SRS encoding process identified the texture and grouped these objects together. However, because the geometric analogy task was modeled earlier, it used a simpler approach, in which the lines of texture were included as part of the object. In this case, the model inspected the object's interior to determine that it contained a texture. The resulting representation was the same in either case.

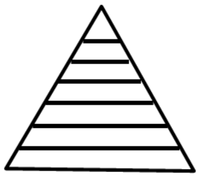


Figure 5.2 Each internal edge of a textured object was added as a separate object.

I manually constructed sketch lattices in CogSketch to support each task. For example, the oddity task required a single sketch lattice for the 2x3 array of images. In contrast, Raven's Matrices required two sketch lattices: one for the matrix, and one for the list of possible answers. I imported the PowerPoint images into CogSketch and placed them into each cell of the sketch lattice. The exact location within each cell of the lattice is unimportant—SRS encodes only an object's location relative to the other objects in an image.

All routines use the following *Perceive* parameters to produce an image's initial representation:

(:focus :groups :group-edges? t :textures? t :shape-features (:fill-colors :border-colors :border-width :relative-size :symmetry :other-attributes) **:shape-trans-relations? t :space-glyphs? t**)

There are a couple things to note here. Firstly, the initial representation is always at the group level. The object and edge levels are used only when necessary for solving a problem. Secondly, the routines use the **:group-edges?** parameter. This indicates that when two objects intersect and neither is a closed shape, they should be joined together to form a single object. Thus, even though I drew "X" shapes as two separate objects, the encoding process treats these as a single object. This is a very conservative encoding process—it joins edges together into larger objects and joins objects together into groups, producing an initial representation with as few elements as possible. These elements are broken down only as necessary to solve a problem.

The encoding process produces a group-level, orientation-specific representation, using the terms given in Table 3.3. If necessary, it can be updated to produce an edge-level representation using the terms in Tables 3.1-3.2.

5.2 Task Comparison

The geometric analogy and Raven's Matrices tasks share an important element: they are both about identifying differences. In geometric analogy (Figure 5.1A), one compares images A and B to find the differences between them. These differences can then be applied to C to compute the answer image. In Raven's Matrices (Figure 5.1B), one compares the images in each row to see how they change. These changes can be applied to the bottom row to compute the missing image.

In contrast, the oddity task (Figure 5.1C) is about identifying commonalities. One compares an array of images and determines what features are constant across it. Then, one selects the image that most lacks those features.

Thus, the first two tasks appear more similar to each other than they are to the oddity task. One would expect them to rely on common cognitive processes. As the following chapters will reveal, the spatial routines for these tasks share many spatial operations. This provides an opportunity to test the generality of those operations, as well as testing the generality of the representations.

The first two tasks are also more complicated, at least from a problem-solving perspective. To solve geometric analogy, one must compare A and B to produce Δ(A,B), the pattern of variance between them. Then, one might apply Δ(A,B) to C to produce D', an imagined answer image (Chapter 6 discusses other possible strategies). This requires reasoning about an abstract set of differences and an imagined solution image. The Raven's Matrix problems require at least as much effort. In contrast, the oddity task requires storing a single generalization in mind (representing the common features) and comparing individual images against it until a less similar image is found. There are no abstract differences or imagined answers to remember.

Given these differences, it seems likely that geometric analogy and Raven’s Matrices place a greater cognitive load on the problem-solver. They should be more likely to tax the solver’s working memory capacity, particularly when the images are more complex or when more backtracking is required. In contrast, the oddity task is less likely to tax working memory capacity. Of course, this will vary depending on each individual problem’s difficulty. In the upcoming chapters, the importance of working memory is evaluated for each task.

5.3 Parameters and Sensitivity Analysis

The spatial routines rely on two numerical parameters. Note that these parameters are separate from the thresholds used in qualitative encoding (see Chapters 2 and 3). Each parameter relates to similarity scores. Because similarity scores are normalized, the range of similarities is 0 to 1. Thus, each parameter has a value within this range.

Firstly, all three tasks are solved by assigning a similarity score to each answer and checking whether one answer’s score is higher or (for the oddity task) lower than the others. The **gt-threshold** parameter (see section 4.4) is the minimum difference between two similarity scores for which one may be considered greater than the other. Its default value is 0.01. This means that two similarity scores can be quite similar (e.g., .80 vs. .78), and the models will still consider the first score to be greater than the second.

Secondly, geometric analogy and Raven’s Matrices involve comparing images to compute a pattern of variance across the images. There are multiple strategies for computing a pattern of variance, so each task has heuristics for evaluating whether a pattern is *sufficient*, or whether the routine should attempt an alternate strategy; the exact meaning of *sufficient* varies across tasks and will be discussed later. The **min-mapping-score** parameter is the minimum similarity score before a pattern can be considered sufficient. Its default value is 0.8. This means that a strategy must result in at least 80% similarity before it is considered sufficient.

I conducted a sensitivity analysis on each of the two parameters. This consisted of varying the value by .05 increments and determining the point at which there is a change in the results. Note that this is not the point at which the models cease to work—it is simply the point where any of the analysis in the following three chapters produces different results. I varied the value by .01 increments near the critical point.

For **gt-threshold**, the valid range of values is .01 - .04. This is a very narrow range, but it is hardly surprising. If two similarity scores are .05 different or more, it should be obvious that they are not the same. The minimum value of .01 indicates that two similarity scores can be nearly identical and still be distinguishable. The only reason this parameter is even required is that, due to internal details of SME, two conceptually identical mappings can produce similarity scores that are slightly different. A **gt-threshold** of .01 will cause these scores to be treated as identical, as they should be.

For **min-mapping-score**, the valid range is much greater: .67 - .87. Thus, there is a wide range of thresholds that can be used to determine if a strategy's score is sufficient. Note that if a value above .87 is used, the models don't begin making errors. They simply become overly cautious, backtracking and trying alternate strategies in places where a human might think the answer is good enough. However, for values below .67, the models become too generous, accepting results from failed strategies.

5.4 Analyses

In analyzing each simulation, I ask three questions. Firstly, does the model perform as well as typical humans? This is the *Sufficiency* question. If the model is as accurate as human adults, this demonstrates that the model's representations and processes are sufficient for the task.

Secondly, are problems that are hard for the model also hard for people? That is, do people do poorly on the problems where the model fails? This is the *Similarity* question. If the model and people fail on the same problems, this suggests there are similarities in their representations and processes.

Finally, can the model help to explain what makes one problem harder than another? This is the *Explanation* question. This analysis is restricted to problems the model correctly solves. It can be done in two ways. First, the model can encode the number of elements in a representation. This number is useful for evaluating working memory load as a factor: do people perform worse when they must keep track of more elements? Second, the model can identify which problems require certain operations to solve. This is usually done via ablation, temporarily removing the model's ability to perform an operation and checking which problems it can no longer solve. If those problems are difficult for a person, we can conclude that that operation may be important; only people who perform the operation, and perform it well, can solve those problems.

6. Geometric Analogy

We begin with the geometric analogy model. This task presents a tempting target for two reasons. Firstly, proportionate analogy ("A is to B as C is to…?") is one of the best-known analogy tasks. It requires thinking about the relations between A and B, the relations between C and D, and potentially the relations between the A/B relations and the C/D relations. Few would disagree that humans use analogical comparison to perform this task. Thus, it is useful for establishing the processes of analogical comparison and problem-solving. Later chapters show how these same processes perform other tasks whose analogical nature is less apparent.

Secondly, the first computational analogy model, Evan's (1968) ANALOGY, was built to perform geometric analogy. Recently, several researchers have built computational models of this task (Schwering et al., 2009; O'Donoghue, Bohan, & Keane, 2006; Ragni, Schleipen, & Steffenhagen, 2007). Thus, the current approach can be compared against both historical and modern models. As we shall see, this approach is unique in that a) none of the inputs or correspondences are hand-coded, and b) it uses general-purpose representations and processes that work across multiple tasks.

Below, I provide some background on the task. I then explain my computational model, constructed in the Spatial Routine framework. Afterwards, I describe an experiment in which I first gathered human data and then ran the model on 20 geometric analogy problems. I used problems from Evans' original simulation, so that the two models could be compared directly. I analyze the results and show how the model explains human performance. Finally, I compare my approach to other geometric analogy models.

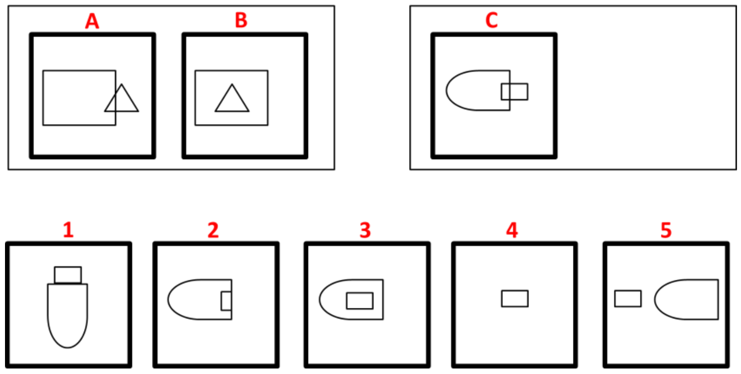


Figure 6.1 A geometric analogy problem.

6.1 Background

Geometric analogy involves comparing images and thinking about differences. However, researchers disagree on which comparisons people make. The debate encompasses two competing strategies. The first strategy involves inserting each possible answer into the analogy to evaluate it. Consider Figure 6.1. The strategy proceeds as follows:

1) Compare A and B to get Δ(A,B), the differences between A and B. In Figure 6.1, there is a change from two overlapping objects to one object inside the other. The rightmost object in A becomes the inner object in B.

2) For each possible answer *i*, compare C to *i* to get Δ(C,*i*), the differences between C and that answer. Then, perform a second-order comparison: measure the similarity of Δ(A,B) to Δ(C,*i*). Whichever answer produces the most similar set of differences is chosen. In Figure 6.1, answer 3 produces an identical set of differences to Δ(A,B), so it is chosen.

The second strategy involves solving directly for the answer:

I) Compare A and B to get Δ(A,B).

II) Compare A and C to get the corresponding objects. In Figure 6.1, the large leftmost shapes correspond, and the small rightmost shapes correspond.

III) Apply the differences in Δ(A,B) to the corresponding objects in C to get D’, a representation of the answer. In Figure 6.1, the small rectangle in C would move inside of the larger shape. Once D’ has been determined, one can compare it to each possible answer and pick the closest match.

Mulholland, Pellegrino, and Glaser (1980) call the first strategy *infer-infer-compare* and the second strategy *infer-map-apply*. However, this assumes that different processes are used to compare images in steps 1), 2), and II). I have argued that structure-mapping can determine differences, identify correspondences, *and* measure similarity. Therefore, I instead call the strategies *second-order comparison* and *visual inference*. *Second-order comparison* refers to the comparison between sets of differences in 2), while *visual inference* refers to inferring the answer image in III).

Evans’ (1968) ANALOGY model was both the first computational model of analogy and an early explanation of how people might perform this task. ANALOGY solved problems via second-order comparison. Given a set of possible answers, it would insert each answer into the analogy and evaluate its quality. See section 6.6 for more details.

Sternberg (1977) studied geometric analogy extensively and concluded that people actually solve it via visual inference. However, Mulholland, Pellegrino, and Glaser (1980) found evidence that people were using second-order comparison. Bethell-Fox, Lohman, and Snow (1984) argued that people may adjust their strategy, depending on a problem’s difficulty. Their eye-tracking data suggested that people typically use visual inference, solving directly for the answer. However, as problems become more difficult, people may abandon this approach, instead simply trying out each possible answer. This behavior appears closer to second-order comparison.

One might conclude that people use visual inference when problems are relatively easy and second-order comparison when problems are relatively difficult. However, this is not the full story. Sternberg (1977) found that participants used visual inference on multiple choice problems, where they had to consider several possible answers. Mulholland, Pellegrino, and Glaser (1980) found that participants used second-order comparison on true/false problems, where they saw a completed analogy and simply determined whether it was correct. The Sternberg problems appear more difficult—at the very least, they put more load on working memory, as there are multiple answers to consider. Why, then, would people use second-order comparison on the easier Mulholland et al. problems?

I believe the answer lies in the algorithmic complexity of the two strategies. This complexity is far easier to measure if one treats all comparisons as equivalent, as predicted by the model. Consider second-order comparison. If there are *n* possible answers, then the number of comparisons is 1 (A to B) + n (C to each answer) + n (Δ(A,B) to Δ(C,*i*) for each answer *i*) = 2n + 1.

Now consider visual inference. If there are *n* answers, the number of comparisons is 1 (A to B) + 1 (A to C) + n (D' to each answer) = n + 2. However, this strategy also requires a non-comparison operation: inferring D' by applying Δ(A,B) to the corresponding objects in C.

Suppose we have a true/false problem. Then the number of answers *n* = 1. The number of comparisons is 3 for both strategies. In this case, one might prefer second-order comparison, as it doesn't requiring inferring D'. This explains why Mulholland et al. found that participants used second-order comparison.

Suppose we have a multiple-choice problem. The number of answers *n* > 1. Now, there will be fewer comparisons for visual inference, and this should be the preferred strategy, as Sternberg found. However, if a problem is particularly complex, participants may be unable to perform the inference operation. In this case, they may fall back on the second-order comparison strategy, which requires only comparison operations.

The model, described below, attempts to solve problems via visual inference. When it fails, either because it is unable to perform the inference or because the inferred image doesn't match any of the answers, the model reverts to second-order comparison. Thus, the model makes concrete predictions about which problems will take longer to solve. These predictions are evaluated in section 6.5.

6.2 Model

The model is implemented in Spatial Routines for Sketches (SRS) (see Appendix B for the full routine). Given a problem such as Figure 6.1, imported into CogSketch, it solves for D through a combination of visual inference and second-order comparison. It begins by comparing A and B with *Find-Differences* to compute Δ(A,B), a pattern of variance between them. It then applies this to image C with *Infer-Image* to produce a representation of the answer image, D'. Recall that this is not an actual image—though SRS can display an approximate image for the benefit of the experimenter, the inferred image representation lacks exact quantitative information.

The model compares D' to each of the answer images. If one answer image is noticeably more similar, and if the answer is *sufficiently* similar, the model picks that answer. An answer will be sufficiently similar if either a) when the images are compared, no differences are found (the **:differences-detected?** result is nil); or b) the similarity score is above the **min-mapping-score** threshold. Note that any time no differences are found, the model immediately returns that answer, without considering further answers.

This strategy will fail if *Infer-Image* is unable to produce D', or if no single answer is the most similar and sufficiently similar to D'. In this case, the model switches to second-order comparison. For each answer *i*, it compares C to *i* with *Find-Differences* to compute Δ(C,*i*), the pattern of variance between them. Then, it compares Δ(A,B) to Δ(C,*i*) to evaluate their similarity. Again, if one answer produces a pattern that is noticeably more similar than the others, it is chosen. In Figure 6.1, answer 3 would be chosen because Δ(C,3) describes a change from (**rightOf** *x y*) to (**contains** *y x*). While a similarity score above **min-mapping-score** is preferred, the model will eventually pick whichever answer produces the best score.

Problem-solving requires making strategic choices. Aside from choosing among the two approaches above, the geometric analogy model must make strategic choices when finding the differences between images, when comparing D' to a possible answer image, and when performing second-order comparison. I describe each of these below.

Computing a Pattern of Variance

The model uses the *Find-Differences* operation to compare two images (e.g., A and B) and compute a pattern of variance between them. Rather than performing only a single comparison, the model implements both basic and complex perceptual reorganization, using a spatial routine And Statement. Recall that perceptual reorganization is when the model re-represents the images so that object shapes will better align during comparison. Figure 6.2A shows an image pair that would trigger basic reorganization, while Figure 6.2B shows a pair that would trigger complex reorganization. I now step through these examples, using the diagram in Figure 6.3.

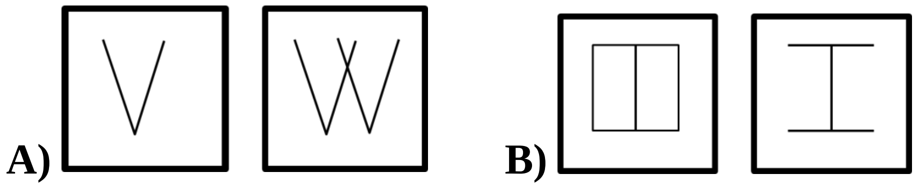


Figure 6.2 Image pairs requiring perceptual reorganization.

In Figure 6.2A, the initial representations contain a single object in each image. The two V's in the second image are combined into a single object because they are open, connected shapes (see section 5.1). When *Find-Differences* is first run (step I in Figure 6.3), it aligns the single V with the double V. This triggers a shape comparison which determines that one shape is a subshape of the other. *Find-Differences*' results indicate that a subshape match exists.

Because there is a subshape match, the routine updates *Find-Differences* (step II), directing it to segment according to the subshape match. *Find-Differences*, in turn, updates the *Perceive* operation that produced the image representation. The double V is segmented into two single V's. The V in the first image now aligns with the leftmost V in the second image (recall that the model resolves ambiguities by aligning objects from left to right). *Find-Differences* returns a pattern of variance indicating that a second, overlapping object is being added to the right of the existing object.

Now consider Figure 6.2B. The initial representations are a square containing a vertical line for the first image, and an "I" shape for the second image. So the first image contains two objects (the square and the line), while the second contains only one. Now, suppose that in the first *Find-Differences* (step I in Figure 6.3), the square is aligned with the "I." Neither object is a subshape of the other, so basic perceptual reorganization (step II) will not trigger.

In step III, the model checks whether there are mismatched objects with common parts. If there are, these objects are broken down into their parts to check whether other combinations of those parts align better. This is done with a Repeat Statement, meaning the model will keep breaking objects down into parts until there are no more candidates. In this case, the square and the "I" have common parts—horizontal edges of similar length. Therefore, the model breaks the square down into four objects, one for each edge. It breaks the "I" down into three objects. After this, there are no more candidates for breaking down, so the Repeat Statement ends.

When *Find-Differences* is called with the parts broken down, the horizontal lines and the center vertical line in the first image align with the lines in the second image. In Step IV, the model checks whether there are objects with the same correspondence patterns that can be grouped together. Here, the three lines can be grouped together because they are connected. Therefore, the model updates *Find-Differences*, again causing *Perceive* to be updated. The three lines making the "I" shape are grouped together in each of the two images. The other two vertical lines in the first image also have the same correspondence pattern—they correspond with nothing. However, they are not connected, so they cannot be grouped together.

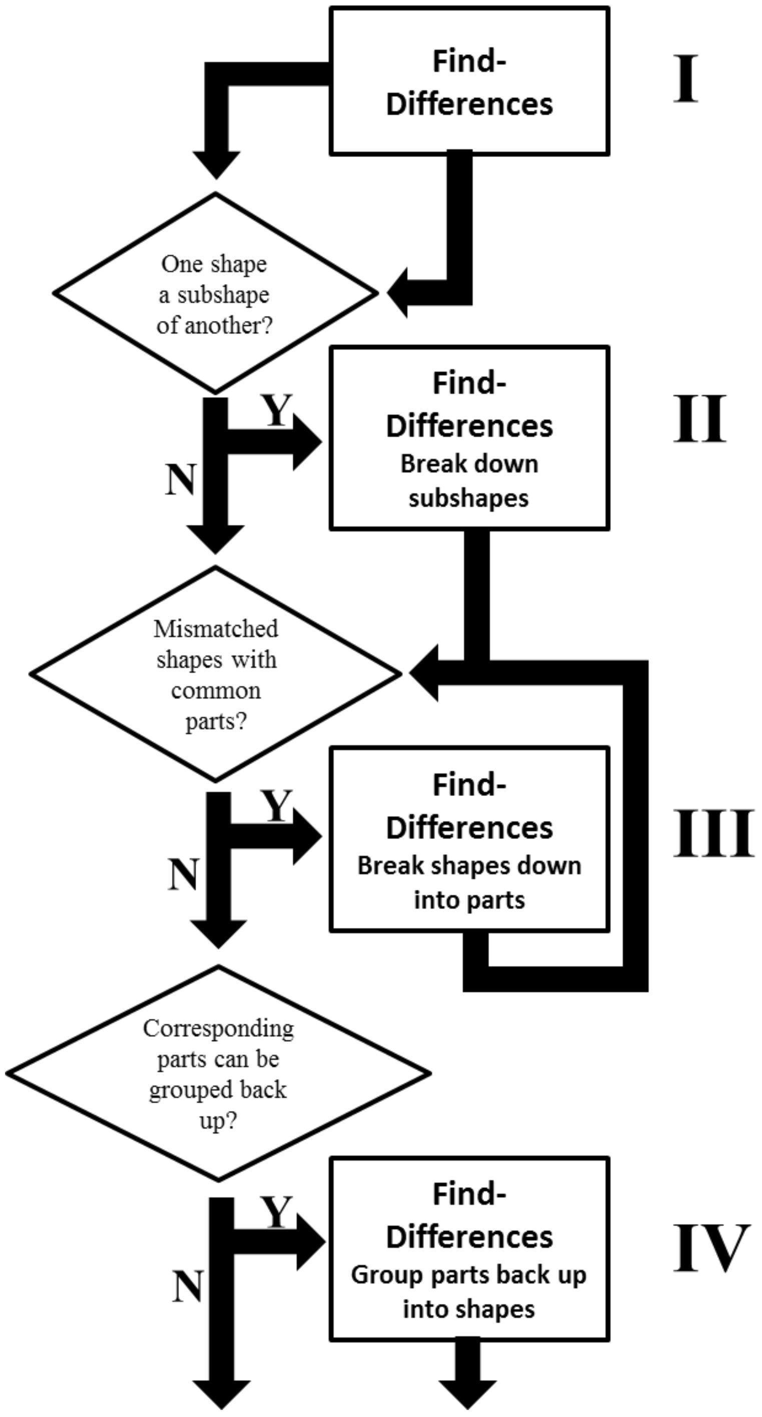

Find-Differences
I
One shape a subshape of another?
Y
N
Find-Differences Break down subshapes
II
Mismatched shapes with common parts?
Y
N
Find-Differences Break shapes down into parts
III
Corresponding parts can be grouped back up?
Y
N
Find-Differences Group parts back up into shapes
IV

Figure 6.3 Strategy for finding differences between two images (see Appendix B for its implementation in the Spatial Routine language).

When *Find-Differences* is run with these final representations, it produces a more coherent pattern of variance: the two outer vertical lines are removed between the first image and the second. Note that complex perceptual reorganization requires more steps than basic perceptual reorganization (Figure 6.2A). Thus, the model predicts that it will be more difficult for humans. Intuitively, Figure 6.2B seems more difficult than Figure 6.2A. I test this prediction empirically in the following chapter, when I analyze the results for Raven's Progressive Matrices.

Comparing D' to Answers

For the visual inference strategy, the model computes D', an image representation, and compares it to each of the possible answers. There is no guarantee that D' will have the same organization as the actual answer image. For example, consider again Figure 6.2A. If the model was computing the second image from the first image, using *Infer-Image*, it would add a second V shape, producing an image with two V shapes. However, the answer image would contain a single object, the double V shape.

Therefore, the model uses basic perceptual reorganization when comparing D' to answer images. This is implemented in a Repeat Statement, similar to the Repeat Statement used for complex reorganization in Figure 6.3. Thus, when comparing images, it will repeatedly segment objects based on subshape matches, but it will never perform complex reorganization, where objects are broken down into their basic parts.

Second-Order Comparison

Figure 6.4 shows several problems that cannot be solved via visual inference. In such cases, the model reverts to second-order comparison. Next, I explain why visual inference fails on each problem. I then discuss the strategic choices made during second-order comparison.

In Figure 6.4A, the pattern of variance between A and B involves a reversal of the **above** relation. There is no **above** relation in C to reverse, so *Infer-Image* fails.

In Figure 6.4B, the pattern between A and B involves a rotation of the square. When *Infer-Image* applies this to C, it rotates the octagon. Then, it compares the rotated octagon back to the original to check whether the desired transformation was achieved. In this case, it was not; the rotation produced an octagon identical to the original one, due to the symmetry in the octagon shape. Recall that in such cases, *Infer-Images* fails, and the model must attempt an alternate strategy (section 4.2).

In Figure 6.4C, *Infer-Images* successfully produces D', the imagined answer. It consists of a large circle. However, D' fails to match any of the possible answers. Therefore, again, the model must revert to second-order comparison.

To perform second-order comparison, the model a) calls *Find-Differences* on A and B to compute Δ(A,B), the pattern of variance between them; b) calls *Find-Differences* on C and each answer to produce Δ(C,answer), and c) compares Δ(A,B) to each answer's Δ(C,answer) to pick the best match. There are two places where strategic choices can be made: when computing Δ(A,B) and when computing Δ(C,answer) for each answer. In each case, the model strategically chooses between four mapping modes:

1) **Normal**: The images are compared as described above (and in Figure 6.3). This produces an initial pattern of variance between them.

2) **Reflection**: This mode changes the preferred transformation when comparing shapes. Recall that identity is normally the preferred transformation. This means that if two shapes are identical, *Compare-Shapes* will not bother checking for rotations are reflections. Here, the model will first check for reflections.

3) **Rotation**: This mode changes the preferred transformation to rotation. For example, in Figure 6.4B, when the model compares images C and 4, it will determine that there is a rotation between them, even though they are also identical.

4) **Alt-Mapping**: The model attempts to find an alternative mapping between the images. This is done by updating *Find-Differences* with the **:try-different-mappings?** directive. Recall that SME can

return up to three mappings. However, mappings will only be returned if their score is within 10% of the top mapping. Thus, this strategy will succeed only if there is a second possible mapping that is nearly as good as the first. For example, in Figure 6.4C, there is a second A->B mapping in which the small triangle in A corresponds to the triangle in B. In this case, Δ(A,B) describes a change in which the larger triangle goes away, while the smaller triangle grows to take its place.

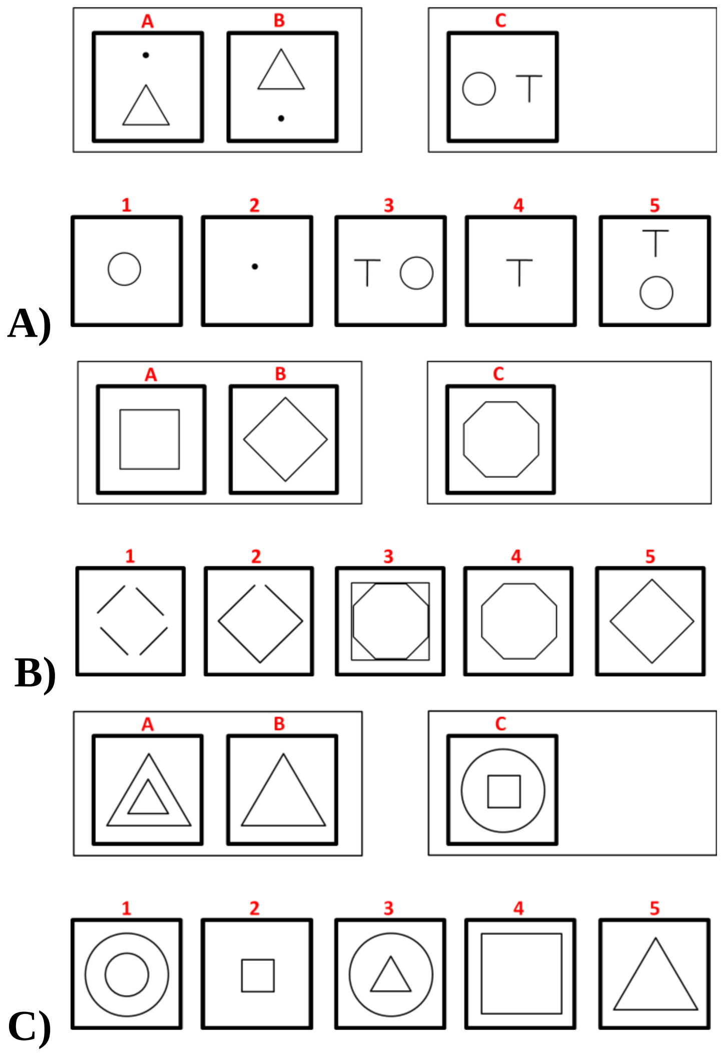


Figure 6.4 Geometric analogy problems requiring second-order comparison.

The model attempts these four modes in the above order. Modes 2) and 3) are only allowed when there is a single object in the images being compared; in complex images with multiple objects, it is cognitively demanding to consider every possible transformation between every corresponding object pair. Alternative mappings (Mode 4) are only considered as a last resort, and only if a valid alternative mapping exists.

The model independently varies the mapping modes for A/B comparisons and C/answer comparisons, beginning with the Normal mode for each. It continues until it meets the sufficiency requirements: one answer's Δ(C,answer) is more similar to Δ(A,B) than the other answers', and either no differences are detected or the score is above **min-mapping-score**. If no answer passes the sufficiency requirement, the model pick's the highest-scoring unique answer across all mapping modes.

Let us consider how second-order comparison handles each problem in Figure 6.4. On Figure 6.4A, the model selects the correct answer using the Normal mapping modes. Δ(A,B) describes a reversal of the **above** relation. Δ(C,4) describes a reversal of the **rightOf** relation. Recall that when two patterns of variance are compared, tiered identicality allows non-identical predicates to align. Here, **above** aligns with **rightOf** because they share a more generalized form: **positional-Generic**. When every expression in the base and target aligns, SME returns no candidate inferences. Thus, the comparison operation indicates that no differences were detected, and so this is a sufficient match.

On Figure 6.4B, the model selects the correct answer when the A/B mapping mode is Normal and the C/answer mapping mode is Rotation. Δ(A,B) describes a 45-degree rotation. Δ(C,4) describes a rotation between the octagons. Since there are several possible rotations between the octagons, it is uncertain which will be returned. However, tiered identicality will allow the A/B and C/4 rotations to align.

On Figure 6.4C, the model selects the correct answer when the A/B mapping mode is Alt-Mapping and the C/answer mapping mode is Normal. Δ(A,B) aligns the small triangle in A with the large triangle in B and describes a change in which the smaller shape grows to replace the larger shape. Δ(C,4) aligns the small square with the large square and describes a similar change.

6.3 Model Predictions

Geometric analogy allows us to test a few predictions of the modeling framework. Firstly, recall that the model's ranking for transformations is identity, canonical reflections, rotations, and non-canonical reflections. That is, when there is both a canonical reflection and a rotation between shapes, the model encodes only the reflection. Figure 6.5 presents an ambiguous problem where the answer depends on whether one sees the "A" shapes as reflected over the x-axis (a canonical reflection) or rotated. If the shapes are reflected, one should select answer 1, as there is a reflection between C and 1. If the shapes are rotated, one should select 2, as there is a rotation between C and 2. The answer participants prefer is discussed below.

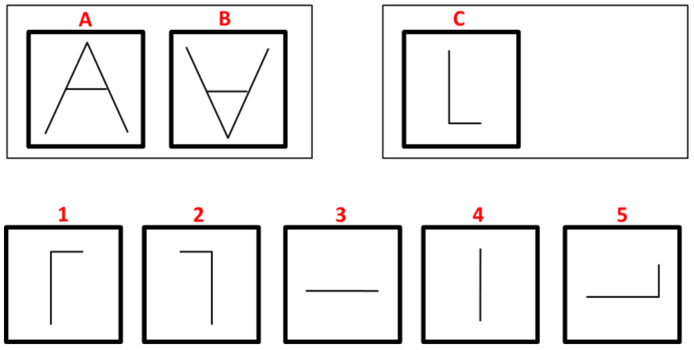


Figure 6.5 The answer chosen depends on whether one prefers canonical reflections or rotations.

Secondly, the model suggests[6] that visual inference will fail when rotating or reflection a shape produces the identical shape, due to symmetry. Thus, assuming canonical reflections are preferred to rotations, visual inference should succeed on Figure 6.5 but fail on Figure 6.6. In the latter case, reflecting the "B" shape will produce an identical "B" shape, which will be treated as a transformation

[6] The previous model (Lovett et al., 2009b), which used only second-order comparison, correctly predicted that problems like Figure 6 would be difficult. The present model was designed with this information in mind, so I cannot claim this is a novel prediction of the model.

failure. If this is true, participants should take longer to solve problems like Figure 6.6 or Figure 6.4B than to solve problems like Figure 6.5.

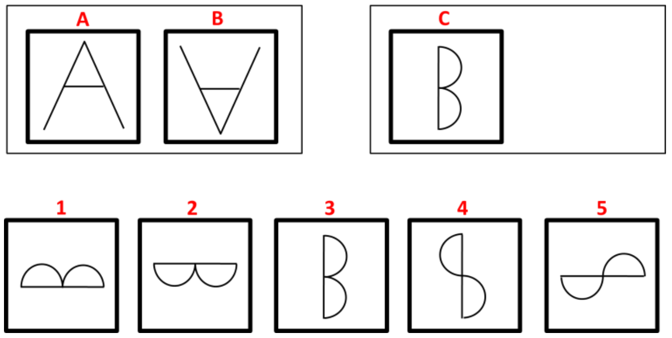


Figure 6.6 Reflecting the "B" shape will produce an identical "B" shape, as in Figure 6.4B.

Finally, during visual inference, a predicate reversal can only be applied if an identical predicate is found. Thus, in Figure 6.4A, visual inference fails because there is no **above** predicate to reverse in image C. However, predicate *removals* do not require that matching predicates be found. In Figure 6.7, the dot removal in A/B results in the removal of a **rightOf** relation, while the dot removal in C results in the removal of an **above** relation. If the model is correct, then Figure 6.7 should be solved faster than Figure 6.4A because Figure 6.7 can be solved via visual inference, while Figure 6.4A cannot.

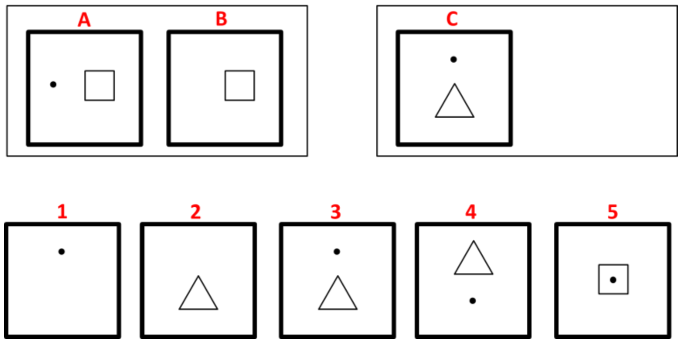


Figure 6.7 The dot to be removed is in different locations in A and C.

6.4 Behavioral Experiment

To evaluate the geometric analogy model, I first gathered human data on the 20 problems from Evans (1968). I reconstructed each problem in PowerPoint. These problems were then both shown to human participants and imported into CogSketch for simulation (see next section). This allowed the model's performance to be compared against human performance on identical stimuli.

Methods

The Evans problems were shown to 34 adult participants. Participants were given a description of the geometric analogy task followed by two simple example problems (without feedback) before they saw the 20 Evans problems. Both the ordering of the problems and the ordering of the five possible answers were randomized across participants.[7]

Before each problem, participants clicked on a fixation point in the center of the screen to indicate readiness. After the problem was presented, participants clicked on the image that they believed best completed the analogy. Participants were instructed to be as quick as possible without sacrificing accuracy. The two measures of interest were the answer chosen and the reaction time, i.e., the time taken to solve the problem.

Results

Figures 6.8-6.14 display the 20 problems along with the mean reaction time and the percentage of participants who chose each answer. Overall, the results show a remarkable degree of consistency across participants. All participants chose the same answer for 9 of the 20 problems, while over 90% chose the same answer for seven additional problems. The greatest disagreement was on Problem 17, on which only 56% of participants chose the same answer, although this was still a statistically significant preference for one answer over the others ($p < .001$). Henceforth, I refer to the answer chosen by the majority of

[7] Because of experimenter error, some participants were given the same random orderings. As many as five participants received one ordering, but on average only 1.5 participants received the same ordering. When one instance of each ordering was randomly selected, the number of participants dropped to 22, and the pattern of results remained the same.

participants as the *preferred answer*. In reporting and analyzing reaction times (including Figures 6.8-6.14), I consider only participants who chose the preferred answer.

The time required to solve a problem varied between 4.5 s and 22.6 s, with a mean of 9.1 s and a median of 7.0 s. Two problems, 10 and 17, required at least 6 s more than any of the others, suggesting qualitative differences in how they were solved. These problems are analyzed below.

I now consider the modeling framework's three previously mentioned predictions.

1) Canonical reflections will be noticed before rotations.

There are two ambiguous analogy problems: 12 and 19. In each case, the answer depends on whether one notices a reflection or a rotation between A and B. On problem 12, 97% of participants chose the answer favoring reflection, while only 3% chose the answer favoring rotation. On problem 19, 82% chose the answer favoring reflection, while 18% chose the answer favoring rotation.

Participants, like the model, appear to notice reflections before rotations. In each case, the reflection is canonical. However, in 12 the reflection is over the y-axis, while in 19, it is over the x-axis. This may explain why reflection was even more dominant in 12. Because image B was directly right of image A, there was a vertical axis of symmetry between the images. Vertical axes of symmetry are particularly salient to people (Corballis & Roldan, 1975; Palmer & Hemenway, 1978). However, even when the pair of images were not symmetric (problem 19), participants heavily favored reflection over rotation.

2) Visual inference will fail when rotating/reflecting a shape produces the identical shape.

Let us consider the problems involving rotation or reflection. On problems 6, 12, 13, and 19, applying the transformation to C produces a new object at a different orientation. On problems 14 and 18, applying the transformation produces an identical object. The model suggests that these latter problems will require more effort because they disrupt the visual inference strategy, forcing the model to fall back on second-order comparison.

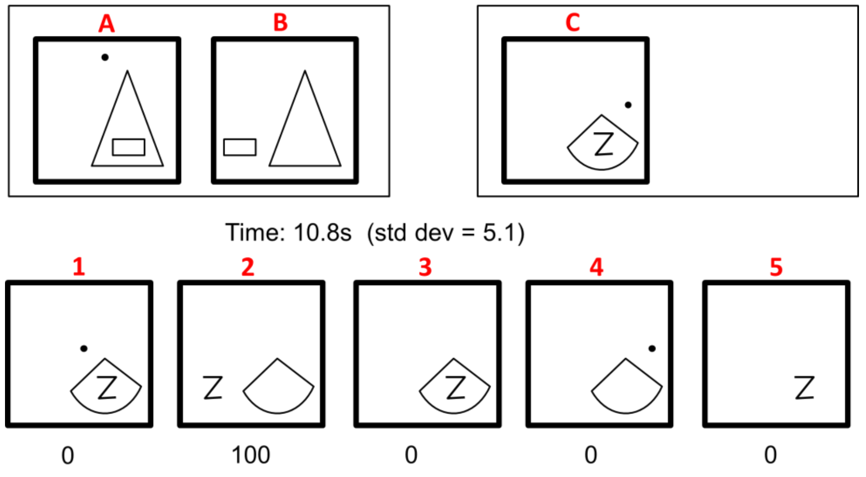
The average time for participants to solve the first set was 6.3 s. The average time for the second set, 10.2 s, was significantly slower (p < .001). This disparity is not the result of a time-accuracy tradeoff. If we exclude the ambiguous problem 18, the proportion of participants who selected the preferred answer was nearly identical on the two sets (95% vs. 93%). Thus, it appears that participants were indeed confused by problems where applying a transformation produced an identical object.

3) Predicate reversal requires identical predicates, while predicate removal does not.

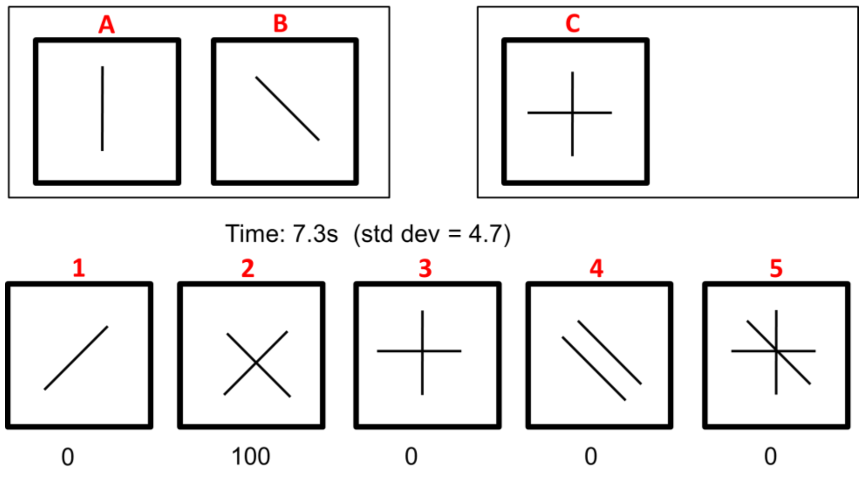
Problem 20 involves generalizing over a reversal of an **above** predicate and a reversal of a **rightOf** predicate. The model is unable to apply a reversal to non-identical predicates, and so visual inference fails on this problem. In contrast, problems 7 and 15 involve removing an object that is in a **rightOf** relation in A but in an **above** relation in C. The model does not require identical or even corresponding predicates for removal, so visual inference succeeds on these problems. Thus, the model predicts that 7 and 15 will be easier to solve than 20.

The average time for participants on 7 and 15 was 6.2 s. The average time on problem 20, 10.8 s, was significantly slower (p < .001). Again, accuracy rates were comparable and near ceiling (99% vs. 97%). Thus, participants appeared to have difficulty when applying a reversal of **above** to a **rightOf** relation.

1



2



3

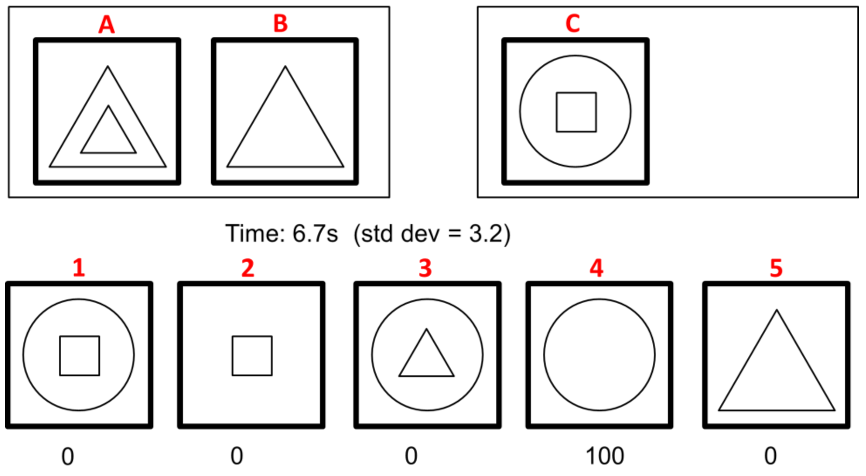
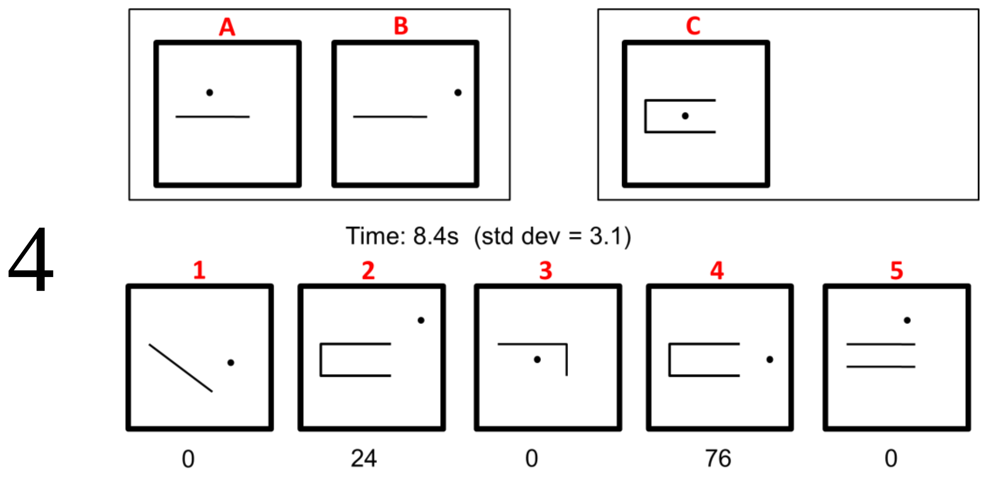
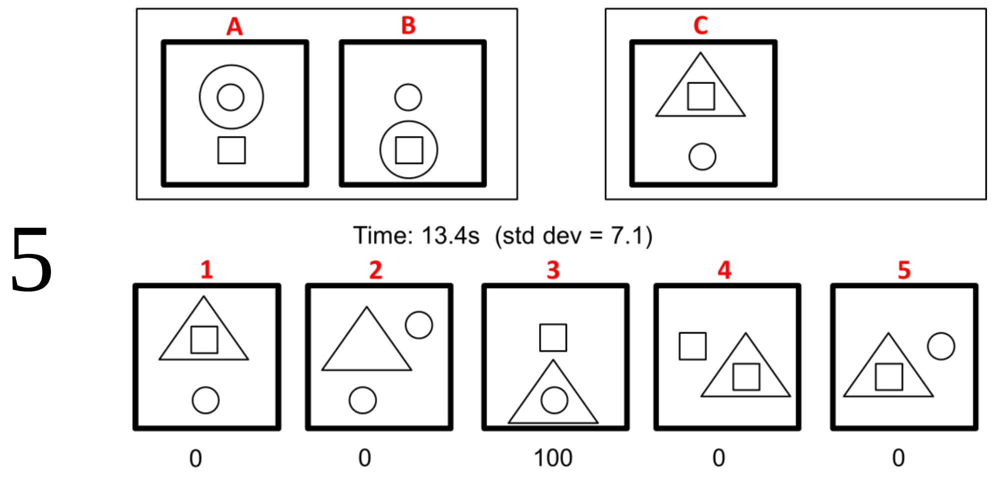


Figure 6.8 Problems 1-3 (times are seconds required for human participants to pick an answer; values below answers are the percentage of participants who picked each answer).





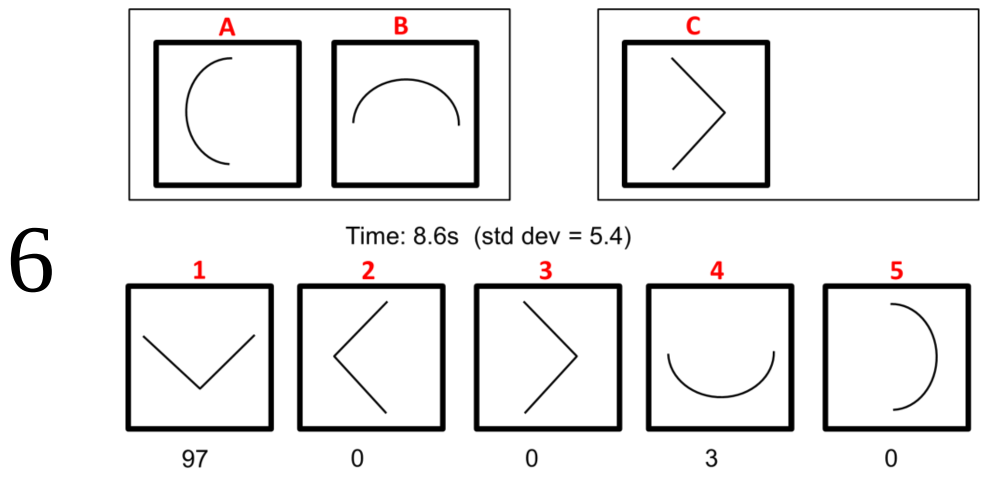
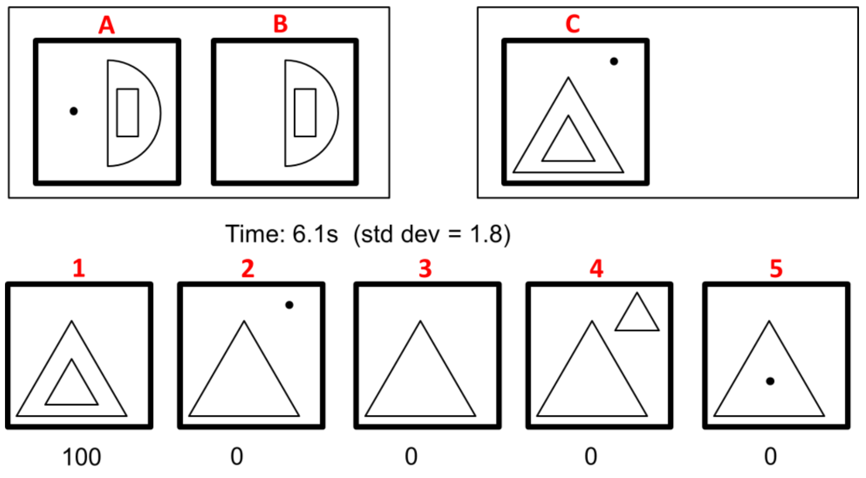
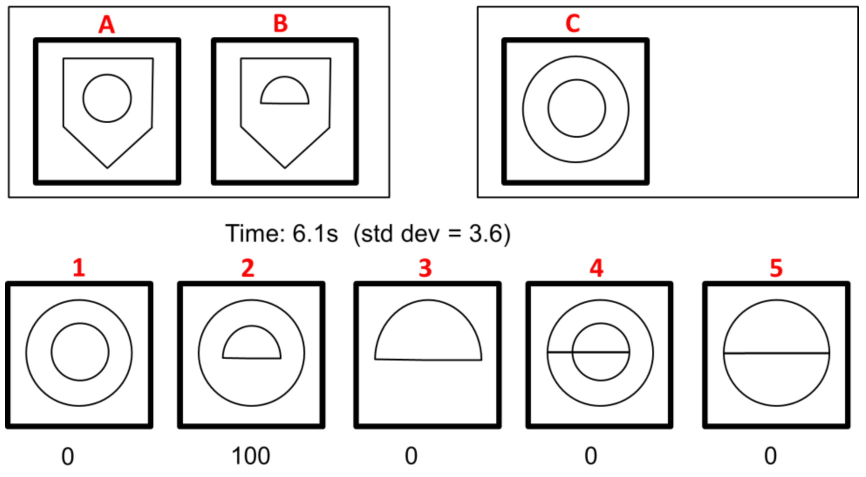


Figure 6.9 Problems 4-6.

7



8



9

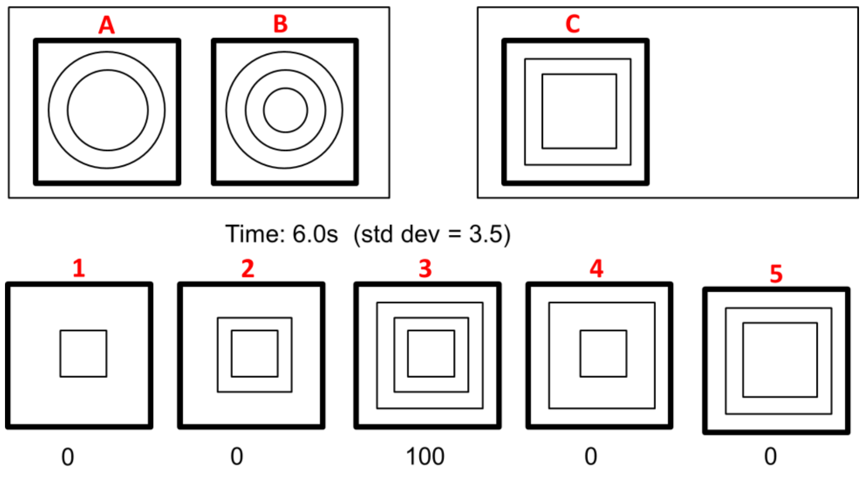
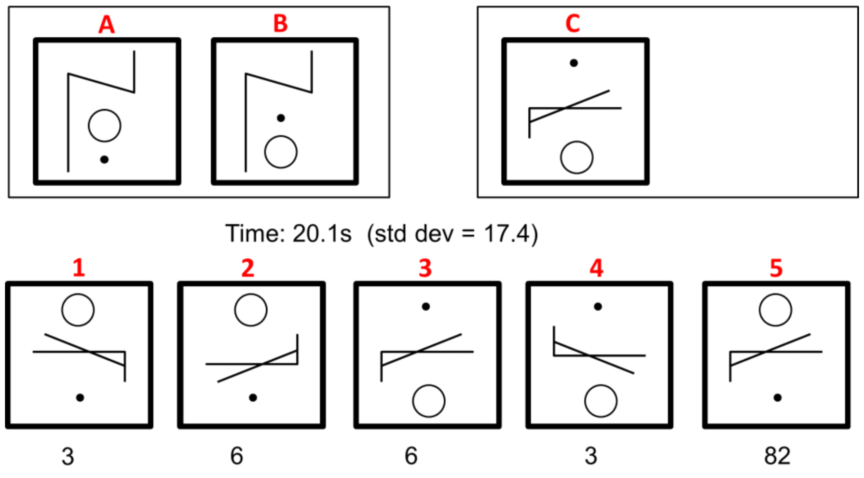
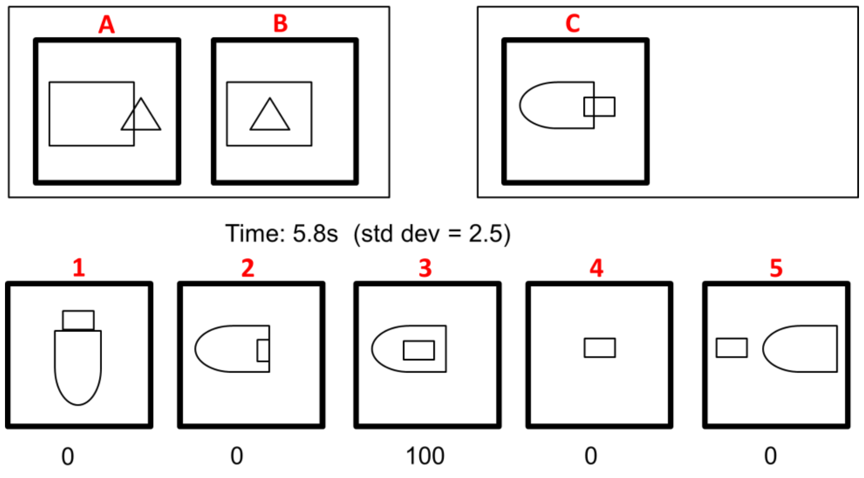


Figure 6.10 Problems 7-9.

10



11



12

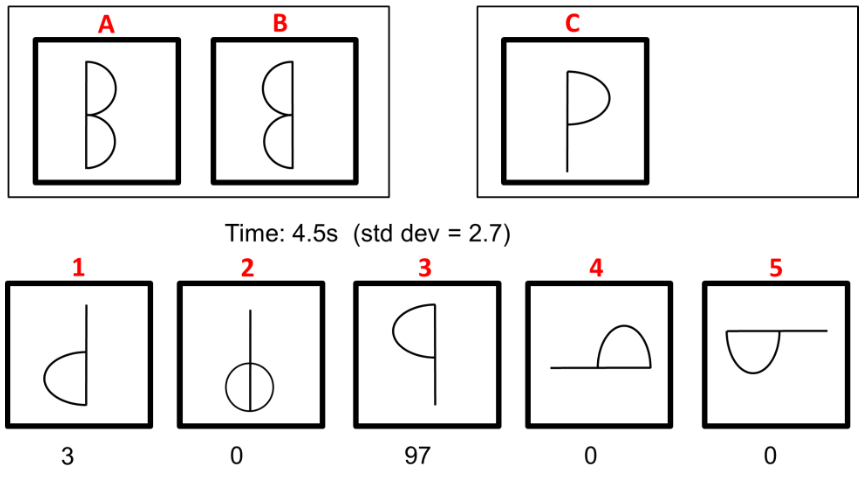
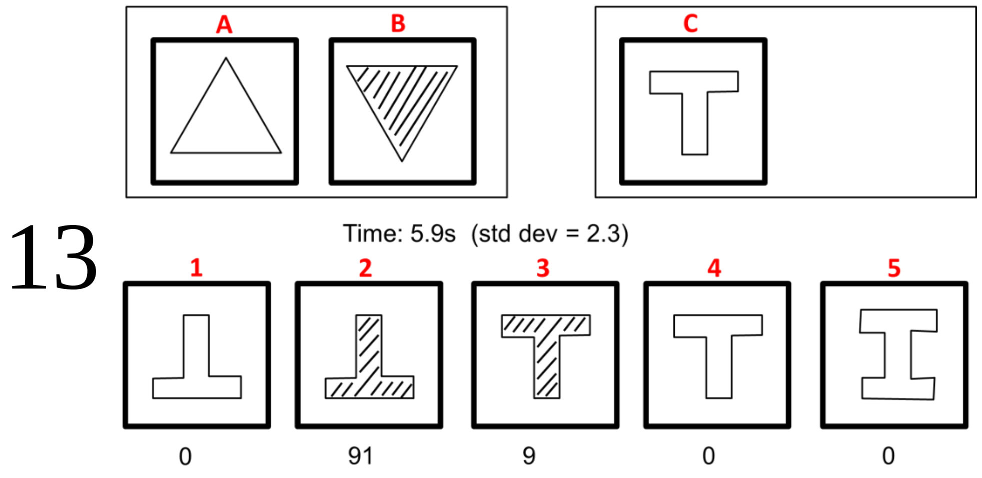
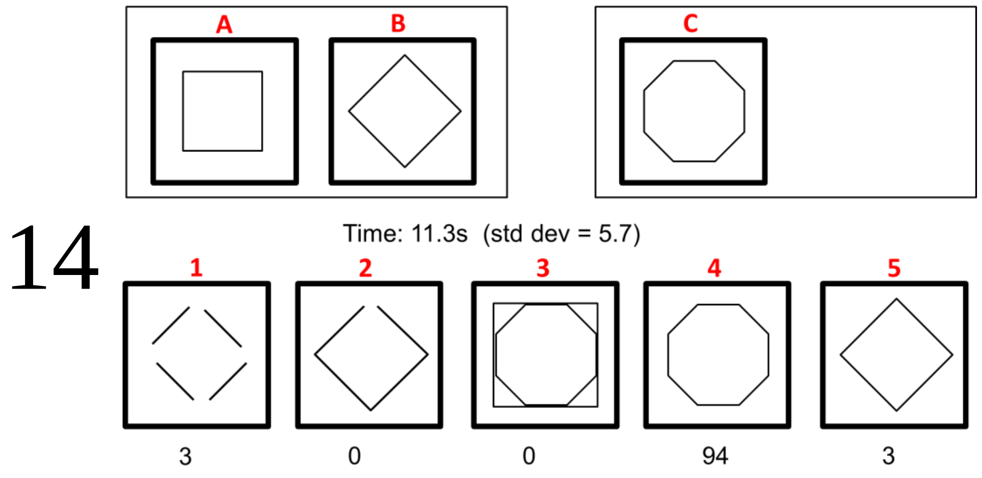


Figure 6.11 Problems 10-12.





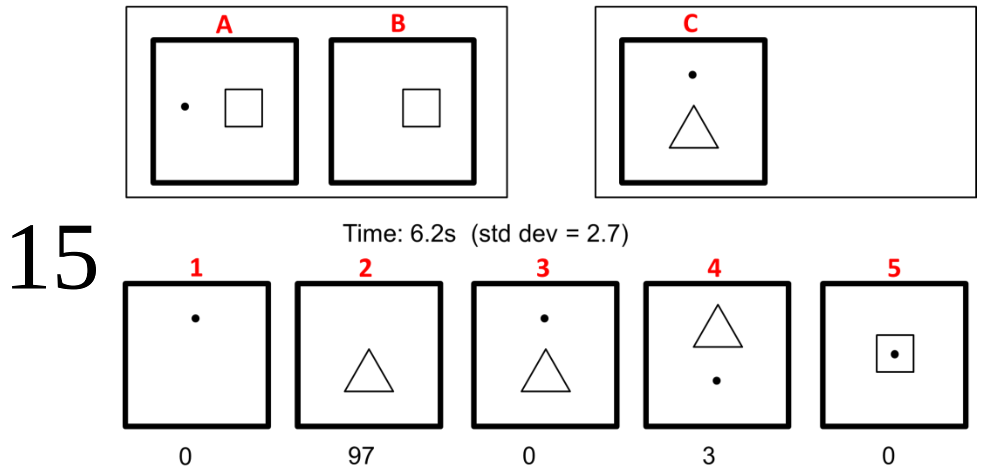
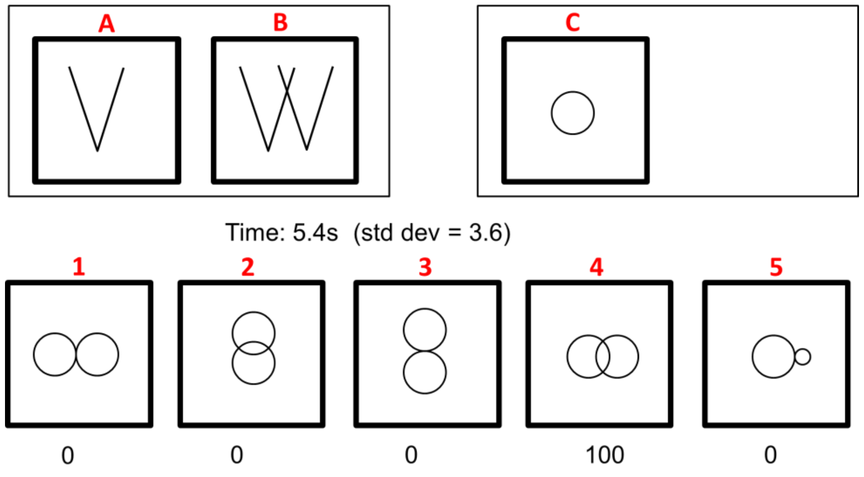
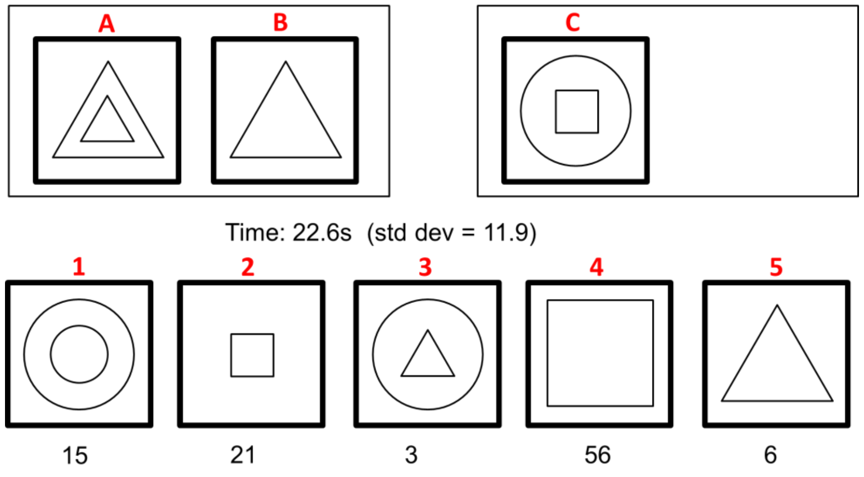


Figure 6.12 Problems 13-15.

16



17



18

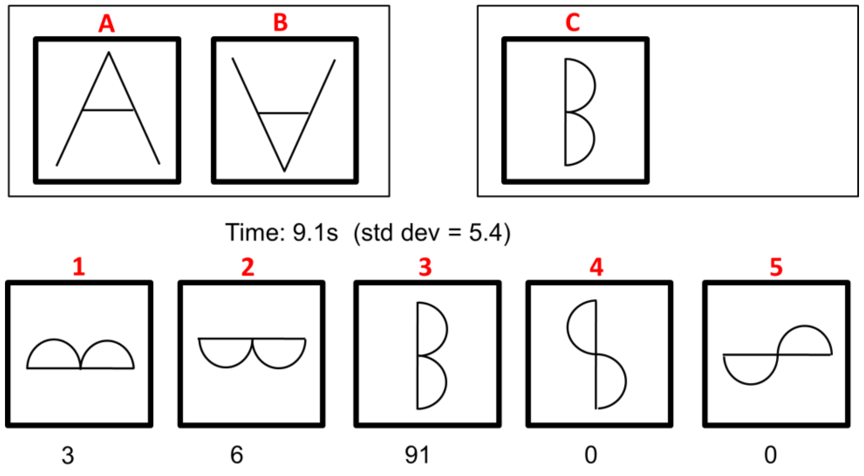


Figure 6.13 Problems 16-18.

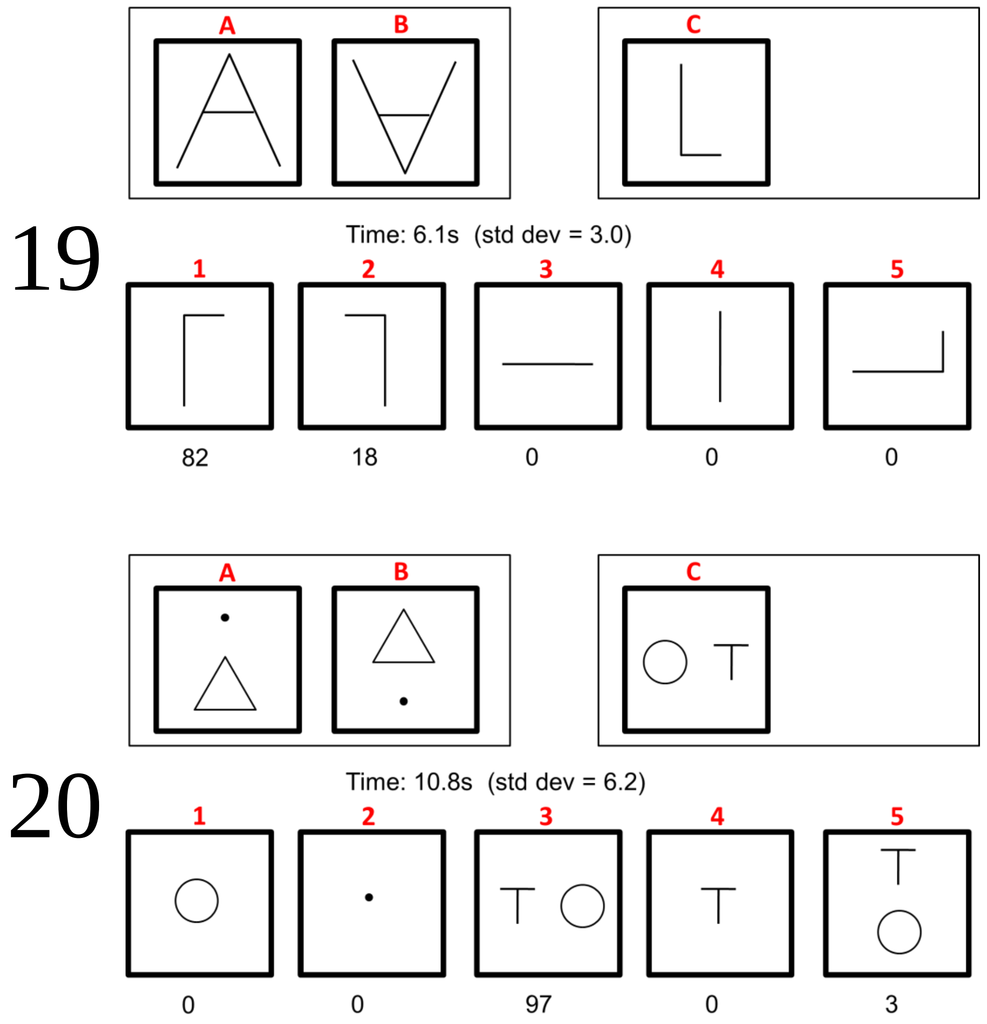


Figure 6.14 Problems 19-20.

6.5 Simulation

I imported the 20 problems from PowerPoint into CogSketch and ran the geometric analogy routine on each problem. I analyzed the results, asking the Sufficiency and Explanation questions.

Sufficiency

The model chose the answer preferred by humans on all 20 problems. Note that these are relatively easy problems—there was over 80% agreement on 18/20 problems, and over 50% agreement on the other two. Given the ease of these problems, the Similarity question was unnecessary—one cannot ask whether problems that were hard for the model were hard for humans because the model solved all problems correctly.

Explanation

In later chapters, I ask whether the model explains human accuracy. However, accuracy on the geometric analogy problems was quite high. Instead, I ask whether the model explains reaction times. Because we are explaining timing, ablation is inappropriate. The issue is not that participants failed on certain problems because they could not perform certain operations. It is that participants took longer on certain problems because they tended to perform certain slower operations. Therefore, rather than using ablation, I code each problem for whether the model performed certain operations while solving that problem.

Below, I consider the factors that might explain what makes a problem difficult for humans. I then describe a linear regression that evaluated the contributions of those factors.

Working memory load

Previous research has shown that geometric analogy problems get harder as either the number of elements or the number of transformations increases (Mulholland, Pellegrino, & Glaser, 1980; Bethell-Fox, Lohman, & Snow, 1984). Mulholland et al. showed that this effect was nonlinear. When the numbers of both elements and transformations increased, the cost to accuracy and reaction time was particularly great. They suggested this was because at some point the problem exceeds people's working memory capacity, requiring a shift in problem-solving strategy.

In my analysis, I focus on the number of elements because this can be objectively determined from the model. The number of transformations is more subjective, depending on what set of changes one considers to be *one* transformation.

There are then two questions: 1) Should we measure the total number of elements, or simply whether the number exceeds some threshold, resulting in a strategic shift? 2) Given that many representations are used to solve a problem, which representation's number of elements should we consider?

I believe there are two possible answers to each question. For 1), we can measure either the total number of elements, or the number above two. I choose *two* because most problems involve 1-2 objects, while a few involve three objects. It may be that only these few problems place enough load on working memory to affect problem-solving. I refer to these possibilities as Total-Elements and Elements-Over-2.

For 2), we could measure either the number of elements in Δ(A,B), the pattern of variance between A and B; or the number of elements in D', the imagined solution to the problem. The elements in Δ(A,B) may be important because Δ(A,B) is a key representation for solving the problem. The elements in D' may be important because the solver must remember this while comparing it to each possible answer. I refer to these possibilities as Diff-Elements and Image-Elements. Note that on problems where the model fails to solve for D', there is no Image-Elements value; in these cases, I use elements in Δ(A,B) for both approaches.

Strategic shifts

The model attempts to solve problems by computing D' directly. If this fails, it makes strategic shifts. These shifts should correspond to reaction time costs for humans. This is reasonable for two reasons. Firstly, a strategic shift involves backtracking and spending more time on the problem. Secondly, strategic shifts place cognitive load on a person's control processes; these processes must monitor the problem-solving, determine that a particular strategy has failed, and suggest a new strategy. Performance may become slower because there is more competition for cognitive resources.

There are three strategic shifts that might slow performance. First, there is the shift to second-order comparison. This is the most basic shift, resulting in a different overall strategy. Second, there is the shift to seeking alternative transformations (rotations and reflections) during *Find-Differences*. Third, there is the shift to seeking alternative mappings during *Find-Differences*. I refer to these shifts as Alt-

Strategy, Alt-Transformation, and Alt-Mapping. Each problem was coded for whether the model performed each strategy shift while solving it.

Linear model

I built a linear model of human reaction times via multiple regression. The explanatory variables were working memory load and the three types of strategy shifts. Because there are four possible working memory measures, I built four models, using each possible measure, and compared the models' R^2 values. R^2 indicates how much variance is explained by the model. An R^2 of 1.0 would indicate the model explained all the variance in human reaction times, while an R^2 of 0 would indicate the model explained none of it.

The best model used the Elements-Over-2/Diff-Elements working memory measure. This model achieved an R^2 of .95. The other three models achieved R^2 values of .88-.85. Thus, the Elements-Over-2/Diff-Elements model appears to best explain human performance. It appears that participants performed equally well when there were one or two elements in a problem. When the number of elements increased beyond two, the problem may have exceeded some people's working memory capacities, causing slow-downs in performance.

Furthermore, it appears that working memory was particularly a factor for $\Delta(A,B)$, the differences between A and B. The number of elements in the solution image was less important. For example, consider problem 9. The solution image contains three concentric squares. However, $\Delta(A,B)$ involves only two elements: a small object is added inside the medium object, while the large object remains unchanged. This problem is easier than problem 1, where all three objects are involved in the change. Thus, working memory appears to be a factor when thinking about and applying *differences*, not when thinking about and comparing images.

Table 6.1 shows the linear model. Overall, this model achieves an R^2 of .95 (.93 adjusted), meaning it explains almost all the variance in human reaction times. The grayed cells indicate which factors made a significant contribution to the model ($p < .01$). The intercept of 6.4 s indicates that the easiest problems took around 6.4 s to complete. Problems with high working memory load required another 5.7 s. Problems involving a shift to second-order comparison required another 4.4 s. Problems involving alternative image mappings required an addition 10.5 s, beyond the cost for second-order comparison.

Intercept	Working Memory Load	Alt-Strategy	Alt-Transformation	Alt-Mapping
6.4 s	5.7 s	4.4 s	- 0.7 s	10.5 s

Table 6.1 Linear model for human reaction times on geometric analogy. Grayed cells indicate significant contributions to the model ($p < .01$).

Note that with correlations, extreme values can result in an overestimation of the explained variance (the R^2 value). In this case, participants took far longer to solve the two problems requiring the Alt-mapping shift (10 and 17). If we remove these data points and rerun the analysis, Alt-Mapping ceases to be a contributing factor, and R^2 drops to .80 (.76 adjusted). Thus, even discounting these difficult problems, the regression explains most of the variance in performance.

The only factor that did not contribute significantly was Alt-Transformation. Alt-Transformation refers to problems like 18, where the model must switch to a Reflection mode to recognize there is a reflection between the two identical "B" shapes. The linear model suggests there is no increased cost for Alt-Transformation problems. However, this does not mean such problems are easy. These problems are difficult in that the model cannot solve them by visual inference. Instead, the model must make the Alt-Strategy shift to solve them, changing to second-order comparison. Once this strategy shift (costing 4.4 s) has been made, there is no additional cost for the Alt-Transformation shift. Problems like 18 are no more difficult than problems like 20.

There are two possible explanations for the above finding. One is that participants make the Alt-Strategy shift and the Alt-Transformation shift simultaneously. The reasoning might be something like "When I try to reflect the 'B' shape, I get an identical 'B' shape. So there's some kind of problem with reflection. I'll try inserting each possible answer into the analogy, but I'll particularly look out for reflections."

Another possibility is that participants *don't* make the Alt-Strategy shift for problems like 18. They solve for D' directly using visual inference. However, the fact that reflecting "B" produces an identical "B" shape causes some confusion, perhaps requiring a strategy shift in how visual inference is performed.

Though it cannot distinguish between the above explanations, the model possesses great explanatory power. Humans take longer to solve every problem where the model shifts to second-order comparison. There is an additional cost for problems where the model must find an alternative image mapping. Together with working memory load, these factors explain nearly all the variance in human reaction times.

6.6 Related Work

Here I describe only models of geometric analogy. More general related work is covered in Chapter 10.

ANALOGY

Evans' (1968) ANALOGY was the first computational model of analogy. Evans tested the model on the 20 problems used here. Operating over lists of points and vertices, it automatically encoded qualitative spatial relations between objects, at least for 11 of the 20 problems[8]. It compared pairs of shapes to find transformations between them. It solved the problems via second-order comparison.

The present model differs from ANALOGY in two important respects. Firstly, it implements both problem-solving strategies: second-order comparison and visual inference. Secondly, it uses a single comparison process, SME, throughout problem-solving. This process is based on a psychological theory

[8] Evans hand-coded the representations for the remaining nine problems. Still, his results are impressive given the technological limitations of the time.

of comparison. In contrast, ANALOGY used slightly different comparison processes for comparing shapes, comparing images, and comparing differences (i.e., second-order comparison). In each case, ANALOGY performed an exhaustive search for the best mapping. Still, ANALOGY was a powerful model that played an important historical role.

Other Models

Recently, several researchers have developed models that use visual inference to solve directly for the answer. However, these models suffer from important limitations. Some use hand-coded symbolic inputs, rather than automatically generating representations. This means the models are unable to reason about quantitative spatial information. For example, they cannot compute shape transformations.

In addition, while most models analogically compare images A and C, none analogically compare A and B to find the A->B transformations. Thus, they miss the key point that the same comparison process can be used throughout spatial problem-solving.

Finally, none of the models have been systematically evaluated on a pre-existing set of problems. Each researcher demonstrates their model only on a small set of problems selected to demonstrate the model. Thus, it is unclear how well the models match human performance.

Schwering et al.'s (2009) model solves analogy problems with ambiguous groupings of objects. Like the present model, it groups objects together based on Gestalt principles, and it can update the groupings to improve the mapping between A and C. The model is limited in that it uses hand-coded symbolic inputs. Also, while it uses an analogical model (HDTP: Gust, Kühnberger, & Schmid, 2005) to compare images A and C, it apparently compares A and B by performing an exhaustive search for the simplest transformation between them.

The LUDI model (O'Donoghue, Bohan, & Keane, 2006) is designed to solve problems involving object attributes (e.g., shading and textures). During the A->C comparison, LUDI first uses structure-mapping to align the relations and then compares the corresponding objects' attributes. Like the above model, LUDI is limited in that it uses hand-coded inputs. In addition, the experimenters hand-code the correspondences between objects in A and B.

O'Donoghue et al. (2006) claim their model is an improvement over ANALOGY and structure-mapping because those models cannot handle object attributes. However, this is incorrect. While ANALOGY and structure mapping focus on relational structure, both approaches encode object attributes also—for example, problem 13 from Evans (1968) involves a change to an object's texture. By including attributes and relations together in a single representation, these approaches are more efficient than the LUDI model. Contrary to the authors' claims, only their very last example could not be solved by ANALOGY or structure-mapping. This final example was designed to be unsolvable by such approaches, as it decouples objects from their attributes. Even LUDI requires extra mechanisms to solve it.

Ragni, Schleipen, and Steffenhagen (2007) developed a model using SRM (Ragni, Knauff, & Nebel, 2006), a system that shifts focus between objects in a grid. Their model automatically encodes symbolic representations from graphical input. It can compute rotations and scale changes between shapes. However, its comparison component is more limited. In fact, the paper is unclear on how the model finds corresponding objects between A and B *or* between A and C. It appears that either this information is hand-coded or the model uses object similarity to identify correspondences.

6.7 Conclusion

This chapter demonstrates that a problem-solving model can be implemented in Spatial Routines for Sketches. The geometric analogy model uses structure-mapping over qualitative representations. Unlike any previous model, it supports multiple problem-solving strategies. It closely matches and helps to explain human performance.

However, thus far we have considered only 20 geometric analogy problems. As the number of problems increases, the range of necessary strategies is also likely to increase. Rather than expanding the model to handle more geometric analogy problems, I now turn to a more complex task: Raven's

Progressive Matrices. In the following chapter, I show that this task can be modeled using a remarkably similar set of representations and processes.

7. Raven's Progressive Matrices

Raven's Progressive Matrices (RPM) (Raven, Raven, & Court, 1998) has been a popular intelligence test for decades. It is successful because it does not rely on static knowledge or verbal ability. Thus, it can be used across cultures and ages to assess *fluid intelligence*, the ability to reason flexibly while solving problems (Cattell, 1963). RPM is one of the best single-test predictors of problem-solving ability: participants who do well on RPM do well on other intelligence tests (e.g., Burke & Bingham, 1969; Zagar, Arbit, & Friedland, 1980; see Raven, Raven, & Court, 2000b, for a review) and do well on other verbal, mathematical, and visual ability tests (Snow, Kyllonen, & Marshalek, 1984; Snow & Lohman, 1989). Thus, RPM appears to tap into core, general-purpose cognitive abilities.

Interestingly, RPM is quite similar to geometric analogy. Consider Figure 7.1. In most RPM problems, the test-taker views a 3x3 matrix of images with the lower right image missing. The test-taker studies the matrix, identifies its underlying pattern, and determines which answer best completes the pattern.

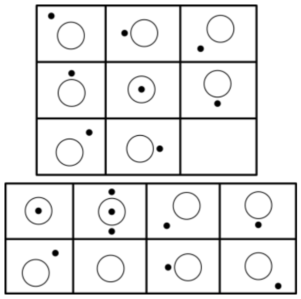


Figure 7.1 A matrix problem. To protect the security of the Raven's Progressive Matrices test, all the Raven's problems presented in this thesis were constructed by the author. Many are analogous to actual problems.

This is essentially the same as geometric analogy, only on a larger scale. In geometric analogy, the relevant pattern describes the change between images A and B. Here, the pattern describes changes between the three images in each row or column. Once those changes are understood, one can apply them to the two images in the bottom row to compute the third image. Alternatively, one can insert each possible answer into the bottom row to test how well it completes the pattern. Thus, as with geometric analogy, a problem can be solved via visual inference or second-order comparison.

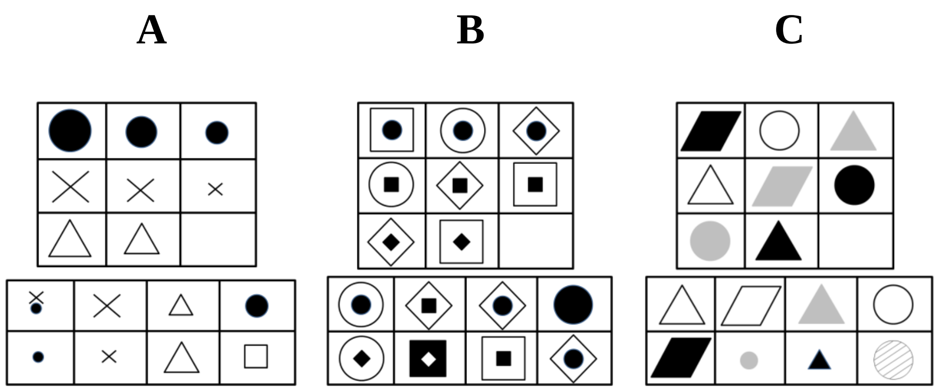
Here, my objective is to demonstrate that a spatial routine similar to the geometric analogy routine can solve RPM problems. This would suggest that the processes being modeled—qualitative encoding, analogical comparison, a spatial hierarchy, and perceptual reorganization—are among those core cognitive abilities that make RPM such a powerful test. Of course, one must also show that the harder RPM problems place more strain on these processes. That is, participants who do better on the test make better use of these processes. This would indicate that the model's processes are not only general-purpose—they are also the keys to successful problem-solving.

I begin with background on Raven's Matrices. I then describe the model, focusing on similarities and differences with the geometric analogy model. While the models are quite similar, RPM requires additional control structures to handle the range of possible strategies. Afterwards, I present a simulation and analyze the model's results. I conclude with related work.

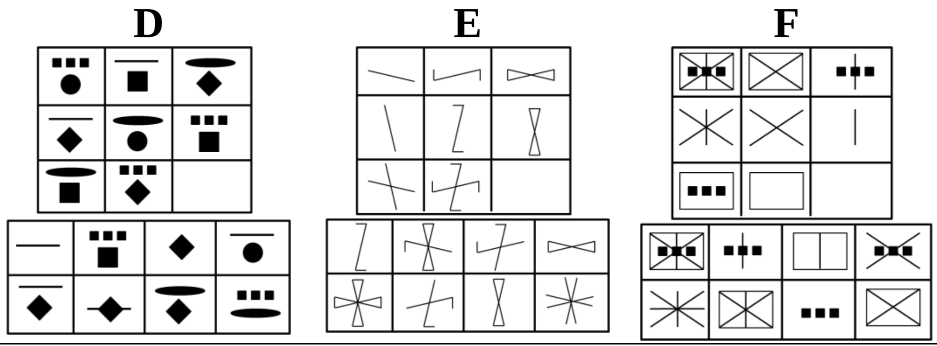
7.1 Background

The Raven's Progressive Matrices test comes in three forms. From easiest to hardest, they are the Colored Progressive Matrices (CPM), the Standard Progressive Matrices (SPM), and the Advanced Progressive Matrices (APM). SPM and APM are similar in many respects. However, APM has a higher distribution of more difficult problems, allowing the test to better differentiate between intelligent individuals.

Carpenter, Just, and Shell (1990) conducted an extensive study of the APM. They ran test-takers with an eye tracker, analyzed the problems, and built two computational models: FAIRAVEN and BETTERAVEN. While FAIRAVEN performed as well as typical college students, BETTERAVEN performed as well as the most advanced students.



Carpenter Rules	Quantitative Pairwise Progression	Distribution of Three, Constant in a Row	Distribution of Three (applies twice)
Find-Differences Strategy	Differences	Literal	Literal + Simple
Answer	3	5	2



Carpenter Rules	Distribution of Three (applies twice)	Figure Addition	Distribution of Two
Find-Differences Strategy	Literal + Simple	Differences	Differences
Answer	4	5	7

Figure 7.2 Example problems for various Carpenter et al. (1990) rules. Also gives the *Find-Differences* strategy the present model would use to solve each problem.

Carpenter et al. found that participants generally solved problems by looking across a row and determining how each object varied between images. Their analysis produced five rules to explain how objects could vary: 1) *constant in a row*: the object stays the same; 2) *quantitative pairwise progression*: the object changes in some way (e.g., rotating or increasing in size) between images (Figure 7.2A); 3) *distribution of three*: there are three different objects in the three images; those objects will be in every row, but their order will vary (Figure 7.2B); 4) *figure subtraction* or *addition*: add or subtract the objects in the first two images to produce the objects in the last (Figure 7.2E); and 5) *distribution of two*: each object is in only two of the three images (Figure 7.2F). While the final rule sounds simple, it is actually quite complex. It requires recognizing that there is no corresponding object in one of the three images.

The FAIRAVEN model implements most of the above rules. Given hand-coded, symbolic representations of each image as input, it analyzes each row via the following steps:

1) Identify corresponding objects. The model uses simple heuristics, such as matching same-shaped objects, or matching leftover objects.

2) For each set of corresponding objects, determine which abstract rules (from above) it instantiates. FAIRAVEN can recognize every rule except the distribution of two.

The model performs this operation on each of the first two rows and then compares the two rows' rules. Finally, it applies the rules to the bottom row to compute the answer. Thus, it implements a visual inference strategy.

The BETTERAVEN model improves on FAIRAVEN in a few ways. Firstly, during correspondence-finding, it can recognize cases where an object is only in two of the three images. This supports rule types 4) and 5). Secondly, it checks for distribution of two rules. Thirdly, it has a more developed goal management system. It compares the rules identified in the top two rows, and if they are dissimilar, it backtracks and looks for alternate rules.

Carpenter et al. argued that this last improvement, better goal management, was what truly set BETTERAVEN apart. They believed that skilled RPM test-takers are adept at goal management, primarily due to a superior work memory capacity. This could explain RPM's strong predictive power. RPM may accurately measure working memory capacity, a key asset in other ability tests. In support of this, other researchers (Vodegel-Matzen, van der Molen, & Dudink, 1994; Embretson, 1998) have found that as the number and complexity of Carpenter rules in a problem increases, the problem becomes harder. Presumably, a greater number of rules places more load on working memory.

However, RPM evaluates more than working memory capacity. Skilled performance depends on perceptual encoding, perceptual reorganization, and correspondence-finding. Carpenter et al. don't consider anything perceptual in their model—they claim that because RPM correlates well with non-visual ability tasks, perception isn't a factor. Their model *does* perform correspondence-finding, and they mention that correspondence-finding could be a source of difficulty, but they do not analyze its effect on human performance.

I turn first to the perception question. I propose that the ability to generate and manipulate perceptual representations may be domain- and even modality-general. In the introduction, I discussed two domain-general problems: entity abstraction and relational abstraction. Entity abstraction involves determining the right level of abstraction for representing a stimulus. For example, one might determine that an image contains several similar objects and group them together. Alternatively, in Figure 7.2C, one must move in the other direction, divided objects up into their individual attributes, and representing each attribute as a separate entity.

Relational abstraction involves noticing complex relations *between* entities. Carpenter et al. discuss one example from their study. Some problems are ambiguous, in that more than one rule could be applied to solve them. For example, imagine a row of arrows pointing in the 12 o'clock, 4 o'clock, and 8 o'clock directions. This could be seen as either quantitative pairwise progression, with the arrows rotating 120° between images, or distribution of three, with three different arrows. Recognizing the rotation requires comparing the arrows and identifying the rotations between them. Simply treating them as three different arrows is far easier.

Carpenter et al. found that participants who recognized the progressive arrow rotations tended to do better overall on the test. They suggested that higher-performing participants were simply more systematic, attempting the pairwise progression rule before the distribution of three rule. However, an alternative possibility is that higher-performing participants were better at relational abstraction—they were more apt to notice how shapes related to each other. Even on an ambiguous problem where either approach works, relational abstraction is helpful because it produces a more abstract, more concise representation—the individual doesn't have to remember three separate shapes.

A more abstract representation places less load on working memory and may improve performance on cognitively demanding tasks. Thus, tasks like RPM may depend on a combination of working memory capacity and abstraction ability. Carpenter et al. mention this point in reference to the Towers of Hanoi, a very different task that correlates highly with performance on RPM.

Sometimes abstraction is *required* to solve a problem. Complex perceptual reorganization depends on a combination of entity and relational abstraction. For example, consider Figure 7.3, which is analogous to SPM problem E9. According to the Spatial Routines approach, one would solve this by a) recognizing that corresponding objects have some common parts (relational abstraction); b) breaking the objects down into their parts, i.e., the individual edges (entity abstraction); c) finding the corresponding edges; and d) grouping the corresponding edges together (entity abstraction). Eventually one would recognize that each row's third image contains the intersection of the previous two images, i.e., the edges that are found in both those images.

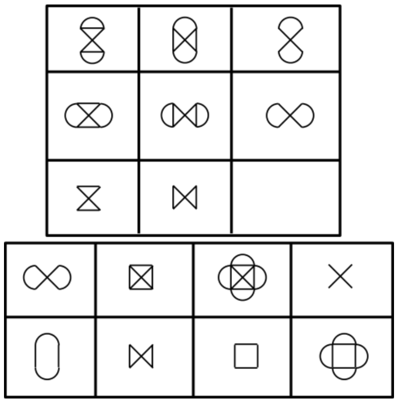


Figure 7.3 This matrix problem requires complex perceptual reorganization to solve. The answer is 4.

Thus, abstraction appears to be an important skill for RPM. What determines a person's abstraction ability? If we consider relational abstraction, detecting a relation between two shapes requires: a) the cognitive motivation to compare shapes; b) sufficiently rich shape representations to support the comparison; and c) a robust comparison process.

The first of these may relate to *need for cognition*. Need for cognition is an individual's intrinsic motivation to seek out structure and meaning (Cohen, Stotland, & Wolfe, 1955; Thompson, Chaiken, & Hazlewood, 1993). It is a domain-general desire to apply cognitive effort to understanding a situation. Individuals who have higher need for cognition may be more inclined to compare objects and identify relations between their shapes. Therefore, abstraction ability may not be specific to the visual domain.

My analysis (see below) cannot distinguish between a desire to compare and an ability to compare. However, I consider perceptual reorganization as a whole, evaluating whether it is a source of difficulty.

Correspondence-finding may also be an important factor. Carpenter et al. mention that correspondence-finding is more difficult for the distribution of two rule because an object is only in two of the three images. However, correspondence-finding may be particularly difficult when an object is missing from the middle of the three images, e.g., the squares in Figure 7.2F's top row. In my analysis, I consider whether this type of correspondence-finding is a source of difficulty.

In the following section, I present the model for solving RPM. It uses the same representation and comparison processes as the geometric analogy model. Like that model, it supports two overall strategies: visual inference and second-order comparison. Carpenter et al. found that their best-performing participants used visual inference, while lower performers may have relied on second-order comparison. In my analysis, I also ask whether reverting to second-order comparison is a source of difficulty.

7.2 Model

Most RPM problems are 3x3 matrices like those shown above. The model solves these by analyzing the top two rows, computing a pattern of variance to describe them, and applying this pattern to the bottom row to infer the answer image. If this approach fails, it reverts to second-order comparison, inserting each answer into the bottom row and checking whether this produces an appropriate pattern of variance. See Appendix C for the full spatial routine implementation.

There are a couple key differences between RPM and last chapter's geometric analogy problems. Firstly, the RPM problems are more difficult, requiring a greater range of strategies. Secondly, the problems contain more information. After computing a pattern of variance for the top row, one can use the second row to evaluate the pattern. Thus, unlike with geometric analogy, one can evaluate one's strategic choices before even considering the possible answers.

The model solves problems as follows:

1) Compute a pattern of variance for the top row.

2) Compute a pattern of variance for the second row.

3) Compare the two rows, computing a generalized pattern of variance for the matrix.

4) If the two rows aren't sufficiently similar, go back and try a different strategy on each row.

5) If they are (or if we're out of strategies), apply this pattern to the bottom row to infer the final image.

6) If visual inference fails, switch to second-order comparison.

The above strategy handles 3x3 matrices. However, the first two sections of SPM use 1x1 and 2x2 matrices. The model handles 2x2 matrices similarly, skipping over step 4) because there is no middle row. There is no way to evaluate the row strategies, but the initial strategies are generally sufficient for these simpler problems.

Some 1x1 problems require an alternate strategy: texture completion. This strategy uses the *Detect-Texture* operation, described in Chapter 4. Thus, the spatial routine for RPM encompasses three possible strategies: texture completion, visual inference, and second-order comparison.

In the following section, I describe 1x1 problems in greater detail. I then discuss the subroutine for computing a row's pattern of variance. As with geometric analogy, this subroutine implements perceptual reorganization; however, there are other strategic choices to consider when comparing three images. Afterwards, I cover the strategic shifts the model makes when comparing the top two rows. I then summarize the strategic shift from visual inference to second-order comparison—this and all later portions of the model are essentially identical to geometric analogy.

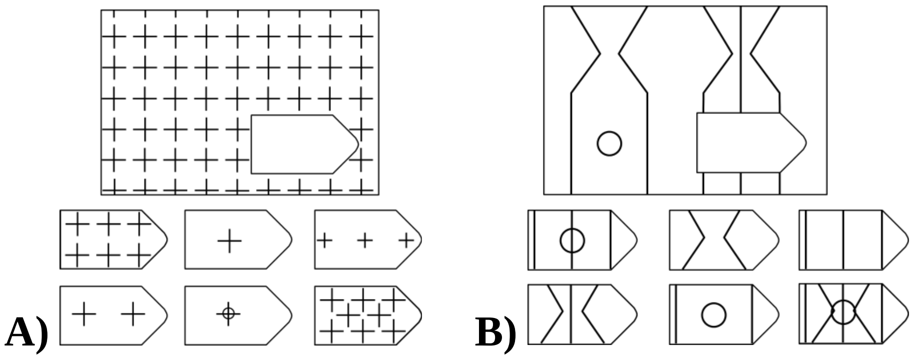


Figure 7.4 Two 1x1 matrix problems.

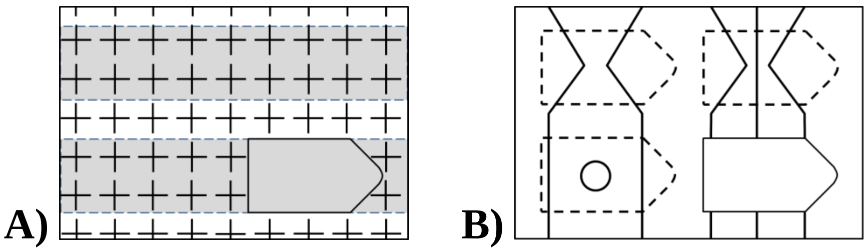


Figure 7.5 Solution strategies for the above problems.

1x1 Matrix Problems

Given a 1x1 matrix problem (e.g., Figure 7.4A), the model first attempts a texture completion strategy, using the *Detect-Texture* operation, as described in section 4.2. Unlike other operations, *Detect-Texture* is purely quantitative—its prints the image to a bitmap, losing all information about separate objects. Thus, it models low-level texture detection. Here, it locates a corridor along the top half of the image (Figure 7.5A) and scans along the corridor, using autocorrelation to detect a repeating pattern. If it finds one, it returns the window size of the pattern.

If a repeating pattern is found, the routine checks whether each possible answer completes the texture. It does this by inserting the answer into the hole in the 1x1 matrix and running *Detect-Texture* on the lower corridor to see if this creates a repeating pattern with the same window size. In Figure 7.4A, the first answer successfully completes the pattern.

Other 1x1 problems (e.g., Figure 7.4B) lack such a repeating pattern. When texture completion fails, the model turns the 1x1 matrix into a 2x2 matrix and solves via visual inference. This process is also described in section 4.4. Using the hole in the 1x1 matrix as a referent, *Locate* identifies three other locations (Figure 7.5B). The problem can then be solved as a 2x2 matrix. Note that 2x2 matrices are particularly similar to geometric analogy, where the top row is images A and B, and the bottom row is images C and D.

Computing a Pattern of Variance

Patterns of variance in RPM are more complex than in geometric analogy—instead of two images, most matrices have rows of three images. Thus, the process involves comparing adjacent pairs of images, identifying corresponding objects across the row, and representing how those objects change. New strategic choices must be considered—for example, should we compare the first and last images, creating a cycle of image comparisons? The full subroutine is given below (Figures 7.7 and 7.8).

Essentially, the subroutine's goal is to produce the simplest, most parsimonious pattern of variance. This is accomplished by endeavoring to meet the following constrains whenever possible:

A) *Identicality*: It's always best if corresponding objects are identical.

B) *Relatability:* If corresponding objects aren't identical, there should be at least some valid transformation or deformation between them.

C) *Correspondence*: Whenever possible, an object in one image should at least correspondence to *something* in another.

If relatability is violated, the pattern of variance must describe a total shape change. If correspondence is violated, the pattern must describe an object addition. It is assumed that either of these makes the pattern more complex and more difficult to store in memory. Below, I refer to violations of relatability or correspondence as *bad shape matches*.

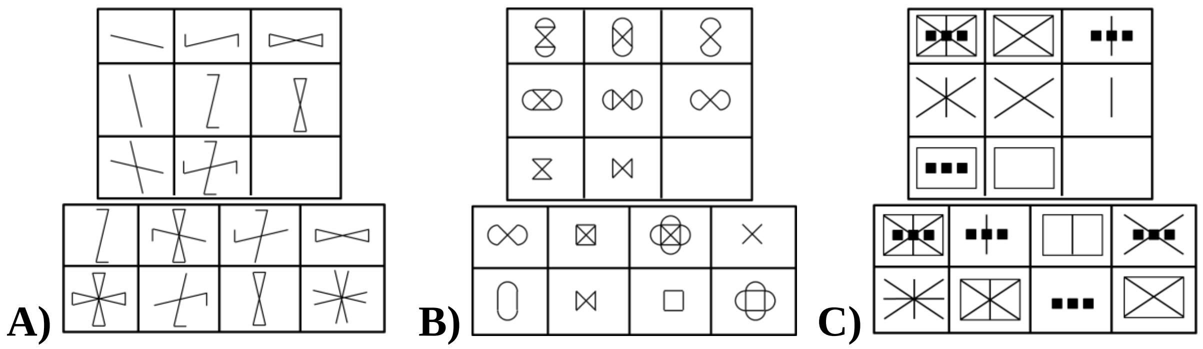


Figure 7.6 Problems where computing the pattern of variance may be difficult.

As with geometric analogy, steps II and III (in Figure 7.7) support basic and complex perceptual reorganization. Basic perceptual reorganization is helpful in Figure 7.6A, a figure addition problem. Because the second image in each row is a subshape of the third image, the third image is segmented into two objects. These objects are identical to the first and second image, allowing the model to detect figure addition.

Basic perceptual reorganization is also used in Figure 7.6C. Consider the leftmost image in the second row. Initially, this appears to be a single object, consisting of three intersecting lines. However, after comparing it to the adjacent image, the model segments the image into two objects: an “X” and a vertical line.

Figure 7.6B depends on complex perceptual reorganization. Consider the top row. In the initial pattern of variance (step I), the corresponding objects are all different shapes. Because they contain similar edges, they are broken down into their edges (step III), with each edge treated as a separate object.

Step IV checks whether there are matching objects in the first and last images of a row. Note that *Find-Differences* only performs this check when there are bad object matches in the first or last image. If there are, it compares the first and last images and checks whether this places any such bad objects into correspondence with identical objects.

In the second row of Figure 7.6C, the vertical lines are bad object matches—the line in the leftmost image aligns with nothing, while the line in the rightmost image aligns with an “X” shape. Therefore, the first and last images are compared, and the model discovers that the lines match perfectly. This triggers step IV, in which the first-to-last comparison is part of the pattern of variance. Step IV includes a directive: the vertical line’s strict shape type is now a constraining strict shape type, meaning vertical lines can only map to other vertical lines. This ensures that the “X” in the second image won’t match the line in the third image. Thus, in the resulting pattern of variance, we have one object (the “X”) found in the first two images, and one object (the line) found in the first and last images.

Note that Step IV also applies to Figure 7.6A. Here, each row’s third image contains a leftover object. When the third and first images are compared, the model discovers a perfect match to this leftover object. Thus, it finds that the third image contains both the objects found in the previous two images.

Step V implements the second half of complex perceptual reorganization: objects with the same correspondence pattern are grouped back together. In the top row of Figure 7.6B, the edges forming the hourglass shape are found in all three images. Thus, these are grouped together to form a single object in each image. Similarly, in the second row, the edges forming the infinity symbol are grouped together. Now each row contains one object that stays the same, along with the following changes: the first image has two vertical lines, while the second image has two horizontal lines, while the third image has neither.

Step VI contains an additional heuristic for improving patterns of variance. If there are mismatched objects (violating relatability) and SME has found a lower-scoring, alternative image mapping that places better-matched objects into correspondence, the model switches to the alternative mapping.

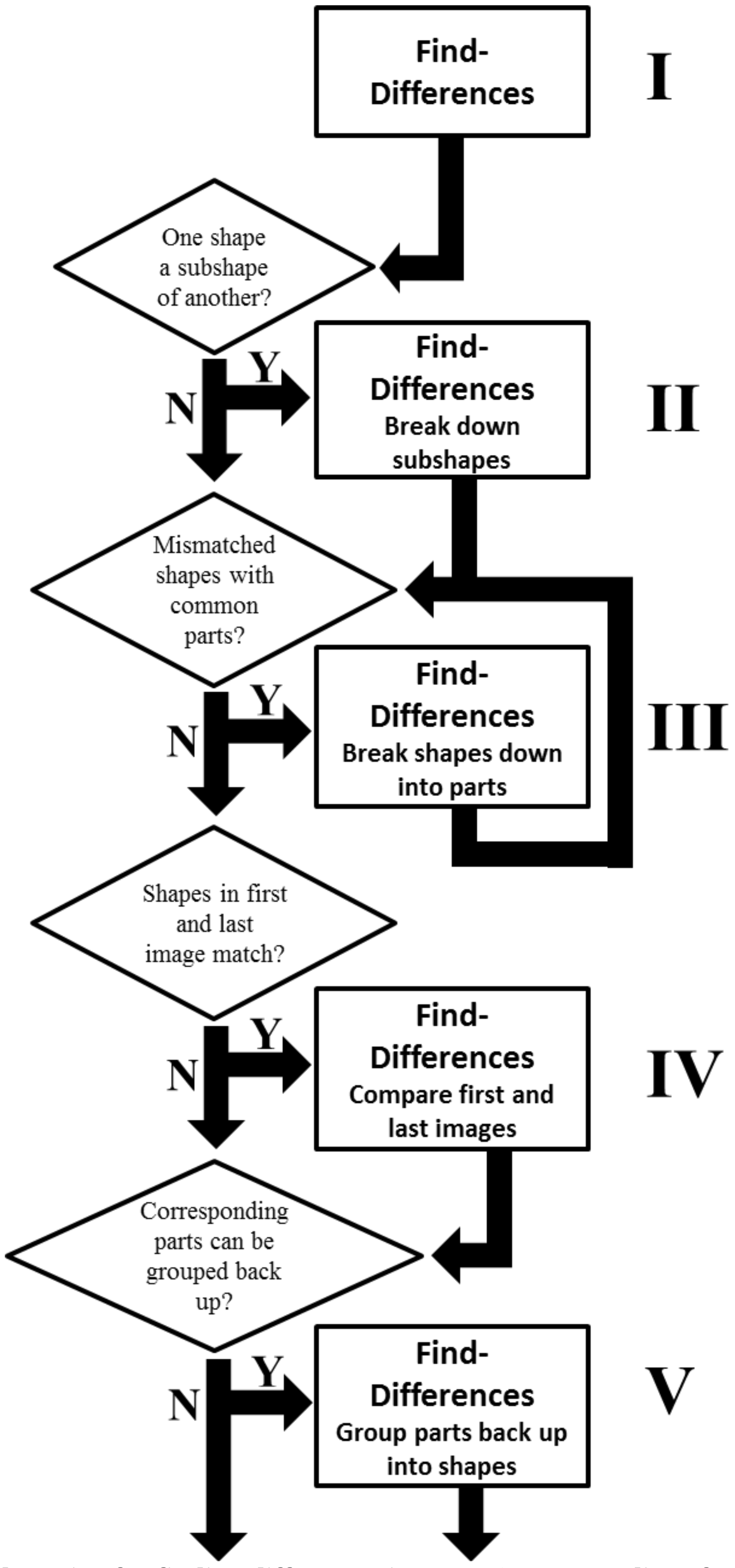

Find-Differences
I
One shape a subshape of another?
Y
N
Find-Differences Break down subshapes
II
Mismatched shapes with common parts?
Y
N
Find-Differences Break shapes down into parts
III
Shapes in first and last image match?
Y
N
Find-Differences Compare first and last images
IV
Corresponding parts can be grouped back up?
Y
N
Find-Differences Group parts back up into shapes
V

Figure 7.7 Subroutine for finding differences in a row (see Appendix C for its implementation in the spatial routine language). Continued in Figure 7.8.

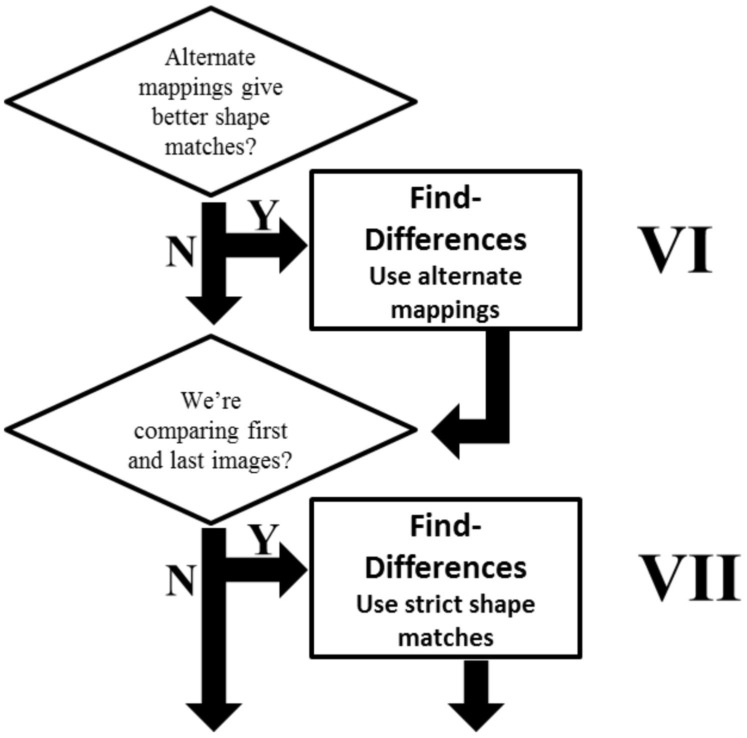


Figure 7.8 Subroutine for finding differences in a row. Continued from Figure 7.7.

Step VII finalizes the first-to-last image strategy. If this strategy was previously implemented in step IV, the model now requires that *every* object only match to other identical objects (i.e., objects with the same strict shape type). This makes the conservative assumption that if we are dealing with complex correspondences (between the first and last images), we will not also be dealing with shape transformations (where non-identical shapes correspond). While this is the correct approach for the SPM, future tests may challenge this assumption.

Representing the Two Rows

There are multiple ways objects can vary in RPM problems. Carpenter et al.'s model has five rule types for describing variation. In contrast, the present model makes two strategic decisions when representing and comparing the top two rows. Firstly, the patterns of variance can be either of two types: *difference* or *literal* (recall the **:pov-type** parameter in section 4.2). The *difference* type is the default—this represents differences between adjacent images in the row.

The *literal* type instead represents what is true in each image, including each object's strict shape type. This type is still based on difference in that if an object or relation is constant across all images, it is not represented here. For example, consider Figure 7.2A. According to the literal pattern of variance, the top row contains three images, one with a square, one with a circle, and one with a diamond. The inner shape is constant across the row, so it is abstracted out of the representation. Whereas order matters in a difference pattern of variance, order can be ignored in a literal pattern of variance. Thus, the first and second rows of Figure 7.2A match perfectly. This strategy is ideal for problems involving Carpenter's distribution of three rule.

Secondly, the patterns of variance can be either regular or *simple*. In a simple pattern of variance, objects are abstracted away from their images. Thus, spatial relations are ignored, and the model considers only how each set of corresponding objects varies. A simple, differences pattern of variance will represent how objects change between images: additions, removals, transformations, deformations. A simple, literal pattern of variance will represent what objects are found in the images. For example, consider Figure 7.2D. The simple, literal pattern of variance states that each row contains a circle, a line, an ellipse, a square, a circle, and a diamond. Because objects are not tied together in images, they are able to vary independently.

The model iterates over the strategies in the following order: normal differences, normal literal, simple literal, simple differences. It evaluates each strategy by building a pattern of variance for the two rows and comparing them with the *Generalize* operation. It only considers strategies that prove *valid*. It stops once a strategy is *sufficient*. If no strategy is sufficient, it picks the highest-scoring valid strategy. The criteria for validity and sufficiency depend on the strategy:

Differences: The default strategy is always valid. It is sufficient if the similarity of the two rows is above **min-mapping-score**.

Literal: This is the most concrete approach, since it represents what holds in each image, rather than differences between images. No leeway is allowed in the comparison. The strategy is valid and sufficient only if there are no differences detect between the rows.

Simple: It is easy for simple patterns of variance to appear similar—because spatial relations are abstracted out, fewer expressions must align to produce a perfect match. Further restrictions must be applied, or simple patterns will often override other, more informative patterns. Thus, they are restricted to only being valid when they produce an otherwise impossible mapping, e.g., one where two top-row objects in the same image map to two middle-row objects in different images.

Note that Carpenter's rules were outcome-based. That is, the model went in already knowing the types of rules it wanted to produce. In contrast, the present model is process-based. The strategic decisions regard the processes that will produce a row representation. Nonetheless, the model can solve problems involving all five Carpenter rules (Figure 7.2), as well as some problems (e.g., Figure 7.3) that don't appear to match any Carpenter rule.

Visual Inference and Second-Order Comparison

After computing a generalized pattern of variance from the top two rows, the models applies the pattern to the bottom row to infer an answer image representation. First, the model builds a pattern of variance for the first two images in the bottom row. It applies any strategic shifts made on the top two rows, e.g., difference vs. literal patterns, simple patterns, or constraining by strict shape class. Next, it generates the answer representation using *Infer-Image*. This process is described in detail in section 4.2.

As with geometric analogy, *Infer-Image* sometimes fails. This might happen for a few reasons:

1) A spatial relation cannot be reversed.

2) A shape transformation cannot be applied. For example, the model wants to apply a part-removal deformation to an object that doesn't have multiple parts.

3) There is insufficient information for perceptual reorganization. For example, consider Figure 7.3. In the top two rows, the objects were reorganized based on what aligned across the three images. In the bottom row, the third image is missing, so the model doesn't trust itself to perform perceptual reorganization. Contrast this with Figure 7.2E. Here, perceptual reorganization affected only the third image in each row. The first two images weren't reorganized. Thus, the model can safely proceed with visual inference, not expecting the unknown third image to affect the organization of the first two.

If visual inference succeeds, the model takes the inferred image representation and finds the best match among the answers. Unlike geometric analogy, the model will not revert to second-order comparison if it fails to find a good match—here, the model is committed to a strategy before it even considers the answers. This is because the 3x3 matrix provides sufficient information to evaluate the model's strategic choices.

If visual inference fails, the model reverts to second-order comparison. It iterates over the answers. For each answer, it inserts it into the bottom row and computes a pattern of variance, again using the strategic decisions from the top two rows. It compares each answer's pattern of variance to the generalized pattern from the above rows. The answer producing the most similar pattern is chosen.

7.3 Behavioral Experiment

RPM has been studied extensively by many researchers. However, I chose to gather new data on participants' performance. I designed a computerized version of the test, allowing access to data not typically available from paper versions, such as reaction times on each problem and answers test-takers considered before making their final answer.

I used the Standard Progressive Matrices (SPM). This test has five 12-problem sections, for a total of 60 problems. Section A uses 1x1 matrices, section B uses 2x2 matrices, and the remaining sections use 3x3 matrices. I was particularly interested in modeling performance on the more difficult 3x3 matrices. Therefore, I only administered sections C-E, a total of 36 problems. It is possible that participants would do worse on these problems without having first practiced on the easier 2x2 matrices. Therefore, I gave participants two 2x2 matrices for practice.

Methods

Participants

The test was administered to 42 Northwestern students. There were 16 males and 26 females. All students ranged from 18 to 22 years old, with a mean age of 18.8 and a median age of 18.

Materials

I ran the experiment in CogSketch. CogSketch has a built-in harness to support behavioral experiments. Within CogSketch, participants viewed scanned-in images of each SPM problem, including the list of possible answers. When they clicked on an answer, a square appeared around that answer. However, participants were still free to consider the chosen answer and potentially change their mind. This addresses a request by Raven, Raven, and Court (2000b), who asked that any computerized version of the test allow participants to view their chosen answer before committing to it.

Overall, the test used the 36 problems from SPM sections C-E, as well as two problems from SPM section B: B3 and B9. The problems from section B were used for practice.

Procedure

Participants began with a brief training period. Instructions within CogSketch described how matrix problems work and told participants that they had up to an hour to finish the problems, but that they might finish in less time. Participants answered two questions from section B of the SPM (B3 and B9). They were given feedback after each response.

Participants then answered 36 problems, sections C-E of the test, in order. Participants answered each problem by clicking on their chosen answer, causing a square to appear around it, and then hitting the “Next” button. CogSketch recorded their final answer for each question, as well as any previously selected answers. Timing information was also recorded. After each question, participants were given the option of taking a break and told to hit “Next” when they were ready to continue.

Results

The mean overall performance was 29.5/36. Two participants (both female) fell more than two standard deviations below the mean. These participants were removed from the analysis, leaving 40 participants. Among these 40, the mean performance was 30.0/36. Splitting by sections, the mean scores were:

Section C: 10.9/12

Section D: 10.7/12

Section E: 8.5/12

An individual with these scores would typically score a perfect 12/12 on the two easier sections (Raven, Raven, & Court, 2000b, Table SPM2), resulting in an overall 54/60. This score is in the 61st percentile for American adults, according to the 1993 norms (Table SPM13).

The mean reaction time (considering only correct responses) was 21.1 s, and the median (by item) was 15.9 s. Note that these problems took far longer to solve than the geometric analogy problems, for which the mean was 9.1 s. Across the 36 problems, there was a remarkably high correlation between mean accuracy and mean reaction time ($r = -.89$). Thus, participants were more likely to fail on problems that took longer to solve.

Discussion

The overall scores in this experiment were slightly lower than expected. Though they were in the 61st percentile from 1993, RPM scores tend to increase over time (Raven, Raven, & Court, 2000b). For example, in 2000 a large-scale examination of Estonians found that 18-year-olds averaged 29.6/36 on the same sections (Lynn, Allik, & Irwing, 2004). One might expect students at a prestigious university to perform better than average 18-year-olds.

It is possible that the participants performed worse because they were taking a computerized version of the test. However, several studies (e.g., Williams & McCord, 2006; Arce-Ferrer & Guzmán, 2009) have suggested that results are comparable on computerized and paper versions. A second possibility is that participants did worse because they lacked the practice from taking the earlier test sections. However, these earlier sections are trivially easy for intelligent adults. I believe a third possibility is more likely: their performance suffered because of the context. Whereas others took SPM as an intelligence test, these participants took it as a psychological experiment for class credit. Thus, they may have been less motivated to put forth cognitive effort. This interpretation is compatible with Williams & McCord's results (2006), wherein undergraduates taking the paper test as a psychological experiment averaged 47-50/60.

Thus, the participants' scores may be slightly deflated due to lack of motivation. However, my interest is in each problem's relative difficulty, rather than the absolute difficulty. If we consider mean performance on each problem, the participants correlated extremely well with the Estonians (r = .96). Thus, I believe the results are a valid dataset for studying relative problem difficulty.

7.4 Simulation

I reconstructed the 60 SPM problems in PowerPoint and then imported them into CogSketch. The final problem (E12) proved difficult to reconstruct, and it requires a strategy outside the bounds of this model (or any other model I know of). Thus, it was counted as an incorrect response and left out of further testing.

I made two changes to assist the model in solving the problems: 1) For section A's 1x1 problems (e.g., Figure 7.4), I labeled the large rectangle as "Problem" and the hole in the image as "Answer." This allowed the spatial routine to locate them by name, rather than using perceptual operations. This seemed acceptable, as the model lacks a more refined human understanding of the problem directions. 2) On one problem, I replaced dashed lines with gray-colored lines. Presently, CogSketch lacks the perceptual ability to recognize dashed lines.

I analyzed the model's results, asking the Sufficiency, Similarity, and Explanation questions.

Sufficiency

The model solved 56/60 problems, placing it in the 75th percentile for the American adults, by the 1993 norms. Admittedly, those norms may be out of date. However, the model also scored higher than the college-age participants, whose performance translated to a 54/60. Thus, the model appears at least as effective as the average adult, and very possibly more effective.

Among the 56 solved problems, the strategy breakdown is as follows:

Texture Completion: 4

Visual Inference: 43

Second-Order Comparison: 9

The model solves most problems using visual inference. Only nine problems require reverting to second-order comparison. Note that four of these are from section A, the 1x1 matrices. Section A problems are conceptually easy, but they sometimes present perceptual problems for the model. In most cases, the model fell back on second-order comparison because of an error in its shape or image representations.

Among the 3x3 matrix problems, the model solved 28 via visual inference and four via second-order comparison. These four problems will be considered in the analysis below.

Similarity

Breaking the score down by section, the model achieved:

A: 12/12

B: 12/12

C: 12/12

D: 10/12 (failed on D11, D12)

E: 10/12 (failed on E11, E12)

The model answered all questions correctly on the easier 1x1 and 2x2 sections. It missed problems in only two sections. Within sections D and E, it missed the final two problems. RPM is designed to get progressively harder, so one would expect the final problems of each section to be the hardest.

According to the 1993 norms, the hardest SPM problems were (from easiest to hardest): D11, C12, E10, E12, D12, E11 (Raven, Raven, & Court, 2000a, Table RS3C3). Thus, the four problems where the model failed were among the six hardest problems. According to the present behavior experiment, the hardest SPM problems were (from easiest to hardest): C12, E9, D11, D12, E10, E12, E11. Thus, the four problems were among the five hardest.

Problems where the model fails are among the hardest for human participants. This supports the argument that the model is using humanlike representations and processes. However, the model correctly solves some problems that are quite difficult for people (C12, E10). The next question will be: can the model explain why those problems are difficult?

Explanation

I begin by discussing C12, an anomalous problem that is surprisingly difficult for people. I then present a series of linear models that converge on an explanation for problem difficulty in RPM.

C12

Problem C12 has is one of the most difficult for human test-takers. This is surprising because it is in one of the easier sections (C is the first to use 3x3 matrices), and there is nothing obviously difficult about the problem. The model solves it using the default strategy (a normal difference pattern of variance, without comparing the first and last images). What makes this problem so challenging for humans?

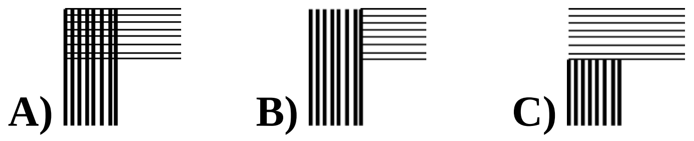


Figure 7.9 Different ways two textures might overlap.

van der Venn and Ellis (2000) have suggested some C problems are difficult due to a conflict between the *perceptual appearance* of the images in the matrix and the answer images. For example, in C12, the first two rows show an object becoming longer. In the bottom row, and only in the bottom row, becoming longer causes the object to overlap another object. Thus, the answer contains a unique perceptual feature: overlapping objects. This might be particularly confusing because the objects are texture patches. It's not entirely clear what overlapping textures *should* look like: are the textures transparent (e.g., Figure 7.9A) or are they opaque, with one occluding the other (Figure 7.9B, Figure 7.9C)?

van der Venn and Ellis characterized several C problems as presenting this difficulty. Why would C12 be particularly difficult? One possibility is the confusion over how textures should overlap. Among the participants, .68 chose the "correct" answer, corresponding to Figure 7.9A, while .03 and .05, respectively, chose the answers corresponding to Figures 7.9B and 7.9C. Thus, while the textures may have contributed to the confusion, participants were not simply unsure how textures should overlap; .24 chose other answers where nothing overlapped.

Can the model provide an alternative explanation? If we consider the 28 3x3 matrix problems solved by visual inference, the model usually infers a representation identical to the answer image. Thus, when it compares its imagined image against the actual image, it scores a perfect 1. In contrast, on C12, the similarity score is only .959. The model fails to infer that there should be overlapping objects because there aren't overlapping objects in the above rows. Thus, the model infers a conflict in perceptual appearance on this problem, but not on other C problems.

The only other problems where the inferred image doesn't perfectly match the answer are E1-E3. Interestingly, these problems are decidedly easier for humans (all mean scores above .90). Why don't they trigger similar confusion for the participants? There are at least three possibilities: 1) The incorrect answers they list may be less confusable with the correct answer. 2) All three problems are instances of the Carpenter figure addition rule, with the third image being the sum of the previous two. This rule may be so intuitive that it overrides any confusion about spatial relations in the answer image. 3) While at least two of these (E2 and E3) introduce a spatial relationship in the third row that wouldn't be inferred from the above rows, it is not a novel relationship. It is a spatial relationship already found within the other images. Only in C12 does the correct answer introduce a spatial relationship (overlapping objects) not found anywhere else in the entire problem. This final possibility best fits van der Venn and Ellis' idea of conflicts in perceptual appearance: participants may be confused when the answer contains a novel spatial relationship not found anywhere in the problem description.

Linear Models

In my further analysis, I focused on the 3x3 matrix problems. I excluded problems the model failed to solve, as well as C12 (discussed above). This left 31 problems: C1-C11, D1-D10, and E1-E10.

I hypothesized that each strategic shift by the model might correspond with increased problem difficulty. I coded problems for whether they required each shift by ablating the model and noting which problems it now failed to solve. My initial factors were:

First-to-Last: Comparing the first and last images in the row.

Basic-Reorg: Basic perceptual reorganization.

Complex-Reorg: Complex perceptual reorganization.

Literal: Using a literal pattern of variance.

Simple: Using a simple pattern of variance.

This allowed me to test whether problems were more difficult when there were complex correspondences (including comparing the first and last images), when there was perceptual reorganization, or when specialized patterns of variance were used. Note that the *literal* pattern of variance is actually more concrete than the default, as it represents what is literally true in each image. In contrast, the *simple* pattern of variance is more abstract, as it breaks an image down into its individual objects.

I modeled participant accuracy via multiple regression. Table 7.1 shows the linear model. Overall, the model explains .67 of the variance in human performance. Thus, this is a strong explanatory model, although far weaker than the geometric analogy model. Four factors contribute significantly to the model: First-to-Last, Basic-Reorg, Complex-Reorg, and Simple. Thus, it appears that problems were more difficult when participants had to compare the first and last images in a row. Problems were also more difficult when participants had to perform perceptual reorganization, breaking each object down into its parts to solve a problem. Finally, problems were more difficult when participants had to use a more abstract, simple pattern of variance.

R^2 (Adj. R^2)	First-to-Last	Basic-Reorg	Complex-Reorg	Literal	Simple	Intercept
.67(.61)	-0.19	-0.11	-0.29	0.07	-0.14	0.97

Table 7.1 Linear model for accuracy of Northwestern students. Grayed cells show significant contributors (p < .05).

This analysis found a cost for both basic and complex perceptual reorganization. However, the cost for complex reorganization was far greater (a drop of .29 vs. a drop of .11 in accuracy). This confirms the model's prediction that complex reorganization is more cognitively demanding.

Finally, problems requiring a literal pattern of variance were trending towards easier, although this factor was nonsignificant. Literal patterns of variance are required for most of section D—this section involves Carpenter's distribution-of-three rule. Looking over these problems, I found that they are actually quite easy: participants averaged over .90 on all but the last two problems (D11, D12), the ones where the model failed. In contrast, problems involving difference patterns of variance (sections C and E) vary far more in their difficulty.

One possible reason for the increased variability is working memory load. It is possible that working memory is less of a factor for literal patterns of variance—recall that in geometric analogy, working memory mattered for the abstract *differences* more than for the individual images. Thus, in RPM, working memory load may be a factor specifically for difference patterns of variances. To test this hypothesis, I replaced Literal with a new factor: Diff-Elements. This factor describes the number of elements in a matrix's generalized pattern of variance. As with geometric analogy, the first two elements aren't counted; thus, this is actually the number beyond two. Furthermore, the number is set to zero any time a literal pattern of variance is used. Thus, this is working memory load for difference patterns of variance only.

R^2 (Adj. R^2)	First-to-Last	Basic-Reorg	Complex-Reorg	Elems2	Simple	Intercept
.80(.76)	-0.18	-0.09	-0.21	-0.05	-0.13	1.01

Table 7.2 Second linear model for accuracy of Northwestern students. Grayed cells show significant contributors (p < .05).

Table 7.2 shows the resulting linear model. This is a much stronger model: it accounts for .80 of the variance in human performance. Furthermore, all factors now contribute significantly. This suggests that working memory is a greater factor when one considers differences between images.

Up until now, I have not considered the final strategic shift: reverting to second-order comparison. Recall that the model makes this shift on four of the 31 problems. In one case (C6), it is unable to apply a shape deformation during visual inference. The other three cases involve perceptual reorganization, either basic (C11) or complex (E8, E9). In each case, the model cannot infer the answer because it is unsure how to organize the objects in the bottom row's first two images.

There are two interesting points to note here. Firstly, the apparent cost for basic reorganization is far higher on C11 (mean accuracy .75) than on other problems (mean accuracy .96, excluding problem E10 which is difficult for other reasons). Thus, Basic-Reorg's effect, already far smaller than Complex-Reorg's, may be exaggerated because it required a strategy shift on C11.

Secondly, problem C11 (.75) is nearly as difficult as problems E8 and E9 (.70, .65), the problems involving complex reorganization. Thus, it may be that any type of reorganization is bad when it disrupts the visual inference strategy.

I built a third linear model, replacing Basic-Reorg and Complex-Reorg with Disruptive-Reorg. This refers to any reorganization that disrupts visual inference, forcing one to revert to second-order comparison. Table 7.3 shows the results. This model accounts for .78 of the variance in human performance. Thus, it performs nearly identically to the above model with one less factor.

R^2 (Adj. R^2)	First-to-Last	Disruptive-Reorg	Elems2	Simple	Intercept
.78(.75)	-0.19	-0.19	-0.05	-0.14	1.01

Table 7.3 Third linear model for accuracy of Northwestern students. Grayed cells show significant contributors (p < .05).

I also built linear models to explain participants' reaction times on the problems (see Table 7.4). These models were quite similar. This is unsurprising, given the high correlation between accuracy and reactions times. There was only one notable difference: Basic-Reorg apparently had no effect on reaction times.

R^2 (Adj. R^2)	First-to-Last	Basic-Reorg	Complex-Reorg	Disruptive-Reorg	Elems2	Simple	Intercept
.80(.76)	11.3 s	2.3 s	8.9 s		4.6 s	8.7 s	9.9 s
.79(.76)	11.5 s			6.7 s	4.7 s	8.9 s	10.0 s

Table 7.4 Linear model for reaction times of Northwestern students (in seconds). Grayed cells show significant contributors (p < .05).

7.5 Discussion

We may draw a few conclusions from the analysis above. Firstly, there is a cost for abstract representations. When participants had to represent *differences* between images, rather than what held true in each image, they became slower and less accurate as the number of objects increased. Participants were also less effective when they had to represent objects separate from their images (simple patterns of variance). This cost might also involve working memory load. It may be more difficult to remember each object when they aren't tied together in images.

Secondly, perceptual reorganization presents a significant challenge. Problems are harder when participants must reorganize the objects in each image, especially when complex reorganization is required. Given the current results, it's unclear whether this cost is incurred during reorganization itself, or during visual inference, when attempting to solve for the answer image.

It's worth noting that problem C6, the other problem that disrupts the model's visual inference, is far easier for the participants (mean accuracy .975). This suggests the model is not telling the full story on visual inference. Indeed, it is possible that many participants solved all the problems with visual inference, without ever requiring second-order comparison. Thus, I suspect perceptual reorganization's cost goes beyond disrupting visual inference.

Finally, the analysis shows a consistent cost for the First-to-Last strategy shift. Recall that this involves: a) comparing the first and last images in the row, and b) constraining mappings to only align identically-shaped objects. This result suggests there is a cost when object correspondences become more complex.

Let us consider the problems requiring the First-to-Last shift: E4, E5, E7, and E10. Going by the Carpenter rules, the first two involve figure subtraction, while the last two involve the distribution of two rule (E7 involves some additional nuances not captured by the Carpenter rules). Does the First-to-Last requirement fully explain what makes these rules difficult?

It appears that it makes some contribution. Carpenter makes no distinction between figure addition and figure subtraction. However, in the present dataset, the figure addition problems (E1, E2, E3, accuracy = .94) are easier than the figure subtraction rules (E4, E5, accuracy = .875). One reason might be that only E4 and E5 require the First-to-Last strategy shift.

On the other hand, First-to-Last doesn't explain why E4 and E5 are easier than E7 (accuracy = .80). The Carpenter rules do, since distribution of two is a trickier rule than figure subtraction. Intuitively, it makes sense that it's easier to encode "subtract the second image from the first image to produce the third image" than to encode "every object is in exactly two, but not three, of the images."

I believe the best conclusion is that, while correspondence-finding is a significant factor, rule clarity is also important. This may also be where learning plays a role in RPM: as test-takers become familiar with a rule, they are better able to seek it out in the future.

In section 7.1, I described Carpenter et al.'s two-step model for finding a pattern in each row. Perhaps a three-step model would be more appropriate:

1) Compare adjacent images, performing perceptual reorganization if necessary. Determine any necessary constraints on correspondences (e.g., only identical shapes match). Identify the corresponding objects.

2) Based on the image mappings, compute the differences between objects.

3) Encode a more abstract rule to describe the differences.

Carpenter et al. noted that participants appeared to discover rules incrementally. They suggested that people attend to one relevant feature at a time, generating a rule to describe how the feature changes. However, according to the present model, individuals would use the full set of features during correspondence-finding and perceptual reorganization. Therefore, I propose that participants may perform steps 1) and 2) a single time, backtracking only if they discover a mistake. Step 3) may be the only consistently incremental step. That is, given a set of correspondences and differences, participants may incrementally consider each set of corresponding objects and attempt to generate a rule describing them.

The present model implements steps 1) and 2), while Carpenter et al. implement a simplified step 1) and step 3). Note that step 3) is not necessary for solving the problems (as the present model shows). However, it is useful for two reasons. Firstly, it produces a more abstract, more compact representation of the row. This may place less load on participants' working memory. Secondly, it can guide the previous two steps. That is, if participants already know which rule to look for, they can compute the correspondences and differences accordingly, much as the Carpenter et al. model does.

The *disadvantage* to step 3) is that if you don't already know all the abstract rules (as Carpenter's model did), it can make the task more difficult. One can get bogged down in searching for a concise rule abstraction. Thus, it pays to not be completely dependent on finding an abstract rule.

This analysis suggests the keys to effective matrix solving are: 1) utilizing rich perceptual representations, 2) reorganizing those representations based on the image mappings, 3) seeking out optimal correspondences between objects, 4) abstracting each row's representation to the point where the two rows align, and 5) recognizing abstract rules that concisely describe what is happening in a row. Of course, these techniques must combine with a strong working memory and effective strategic control during problem-solving.

7.6 Related Work

Hunt (1974) proposed two models for solving a subset of the Advanced Progressive Matrices (APM). While the first was a concrete Gestalt model, the second was an abstract analytic model. The analytic model had a pre-existing set of operations (e.g., constancy, expansion, addition, composition). Given a row or column (and assuming it knew the representations and correspondences), it would check whether each operation described the changes between images. While Hunt's model was never fully implemented, the model's operations and strategies match several of the rules Carpenter et al. would later define.

Carpenter, Just, and Shell (1990) presented the first automated computational models. As described above, their models incrementally identify instances of five hard-coded rule categories. Their FAIRAVEN model performed as well the typical college student on APM, while their BETTERAVEN model performed as well as the best college students. Their models were limited in that they use hand-coded input representations, they possess a simplified correspondence-finding strategy, and they assume prior knowledge of the various rules for Raven's matrices. The models also do not implement perceptual reorganization—a brief note in the paper suggests that the researchers hand-code possible segmentations of the objects for problems where the initial perceptual organization fails.

Rasmussen and Eliasmith (2011) present a spiking neuron model designed to solve the APM. The model relies on the user to hand-code both symbolic image representations and object correspondences between images. Given the appropriate representations, it generates three types of rules describing how objects vary. These rule types are analogous to Carpenter et al.'s five rule types, but each rule generator is implemented using neurons. The authors present no results for their model.

Cirillo and Ström's (2010) model for solving SPM shares several features with my own. Representations are automatically generated from vector graphical descriptions (similar to sketches, but more abstract) and organized hierarchically. The model moves down the hierarchy, going from the most concrete to the most abstract, as needed during problem-solving. However, the model has a couple limitations: 1) It does not perform shape comparison; rather, it recognizes five shape types (e.g., ellipse, rectangle) and encodes the rotation from the upright position. Thus, it cannot handle complex shapes and shape deformations. 2) Rather than building up differences from image comparisons, the model is entirely top-down: it possesses seven pattern-matching strategies, each with its own rules about correspondence-finding (most search exhaustively over all combinations). Overall, the model solves 28/36 problems from SPM sections C, D, and E, suffering particularly in section C, which has more perceptual shape-based problems.

Kunda, McGreggor, and Goel (2010)'s SPM model is perhaps the most perceptual. It operates directly on scanned images of each matrix cell. Rather than generating a symbolic representation, it simply identifies quantitative transformations between images. Unsurprisingly, it suffers due to lacking any abstraction ability. It solves 35/60 problems on the SPM.

It is interesting to note that models of the Advanced Progressive Matrices (Carpenter, Just, & Shell, 1990; Rasmussen & Eliasmith, 2011) use hand-coded image representations, while models of the Standard Progressive Matrices (the present model; Cirillo & Ström, 2010; Kunda, McGreggor, & Goel, 2010) perform at least some automated encoding. I believe one major factor making the APM harder than the SPM is the perceptual encoding—APM problems are more likely to involve entity abstraction, perceptual reorganization, etc. The greatest challenge, a model of APM that automates encoding and reorganization, has yet to be met.

8. The Oddity Task

Thus far, I have simulated studies of American college students only. However, there are important questions about the universality of human spatial cognition. To what extent do representations and processes vary or remain constant across cultures? Dehaene and colleagues (Dehaene, Izard, Pica, & Spelke, 2006) demonstrated that some aspects of spatial reasoning are universal. They used an oddity task (Figure 8.1), in which participants were shown an array of six images and asked to pick the image that did not belong. Solving this task requires sensitivity to two-dimensional geometric concepts such as parallel lines, right angles, and axial symmetry. Dehaene et al. compared the performance of two population groups: North Americans and the Mundurukú, a South American indigenous group. They found that while the Americans performed better than the Mundurukú overall, the Mundurukú performed above chance on nearly all problems. Furthermore, the error patterns correlated across cultures. Thus, the Mundurkú appeared to utilize the same geometric concepts as the Americans, despite having few words for such concepts in their language and no formal schooling in geometry.

These results have been taken as evidence for core knowledge of geometry (Spelke & Kinzler, 2007), an innate, universal cognitive module that deals with geometric concepts. However, the two groups did not perform identically: Mundurukú of all ages performed at about the same level, while Americans tended to improve with age (Newcombe & Uttal, 2006), suggesting that culture-specific learning is a factor. Further research is needed to determine which aspects of spatial cognition are universal and which are learned. Here, I study this question using a Spatial Routine model. By ablating different aspects of the model, I can cause it to perform more like one group or the other, thereby producing novel, testable hypotheses about how spatial cognition varies across cultures.

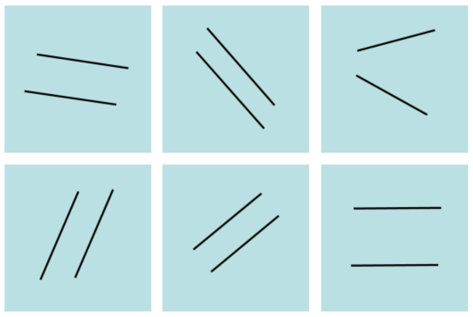


Figure 8.1 Oddity task problem from Dehaene et al. (2006). The image without parallel lines is the odd one out.

8.1 Background

In contrast with the previous tasks, the oddity task is relatively new and has not been studied extensively. In this, the first computational model, I propose that people use structural comparison over hybrid, hierarchical representations, as in the other tasks. The primary distinction is that, whereas those tasks were about identifying differences, this task is about identifying commonalities. For example, in Figure 8.1, most images contain parallel lines. Once this commonality is identified, one can locate the image that lacks this feature.

I propose that differential performance may be linked to cultural differences in encoding of spatial representations. Research suggests spatial representations vary based on exposure to external artifacts such as language (e.g., Loewenstein & Gentner, 1995; Haun et al., 2006; Hermer-Vazquez, Moffet, & Mukhold, 2001). While the oddity task study demonstrated the Mundurukú have many of the same geometric concepts as Americans, there may be differences in when and how they encode those concepts. Here, I make the simplifying assumption of a single, broad set of concepts available to both groups—the spatial representation scheme described in Chapter 3. I then test whether the groups vary in how easily they can use certain concepts. In particular, I look at entity and relational abstraction.

8.2 Model

The model's overall strategy is to *Generalize* over a subset of the images, determining what is constant across them. It then compares each remaining image to the generalization to check whether one image is noticeably less similar. The model uses the rows as subsets. For example, in Figure 8.1, a generalization over the top row would determine that each image contains two straight edges. Each bottom-row image also contains two straight edges, so none would be noticeably less similar. However, a generalization over the bottom row would determine that each image contains two straight, parallel edges. Because the upper right image doesn't contain parallel edges, it will be less similar to this generalization.

An image is *noticeably less similar* if its SME mapping score falls below the other five images' scores—that is the similarity of bottom-row images to the top-row generalization, and the similarity of top-row images to the bottom-row generalization. The model again uses the **gt-threshold** in comparing similarity scores. That is, the difference between the scores must be at least **gt-threshold** for one to be considered lower than the others.

The division by rows is consistent with the Raven's model. However, I make no claims about humans focusing on rows specifically. The model would perform similarly with arbitrarily different subsets.

As with the previous models, this model must make strategic shifts during problem-solving. Most of the shifts relate to representation, as this task doesn't require reasoning about abstract patterns of variance. Specifically, there are three shifts: choosing a level of abstraction, filtering out irrelevant information, and choosing a similarity measure. I describe each strategic shift below. I then show how the model brings these strategic decisions together.

Level of Abstraction

Recall that the model utilizes three levels of abstraction: groups, objects, and edges. I have argued that people typically begin with the highest level (groups) and then become more precise as needed. In the oddity task, many problems can be solved at the group level. For example, in Figure 8.2A, every image contains a single group of dots except the lower right. Figure 8.2B can also be solved at the group level. Recall that a) when no groups are available, group-level representations look like object-level representations; and b) axis-alignment is an edge-level feature that can "bubble up" to the object level. That is, when every edge in an object is axis-aligned, the overall object receives a **2D-Shape-Axis-Aligned** attribute.

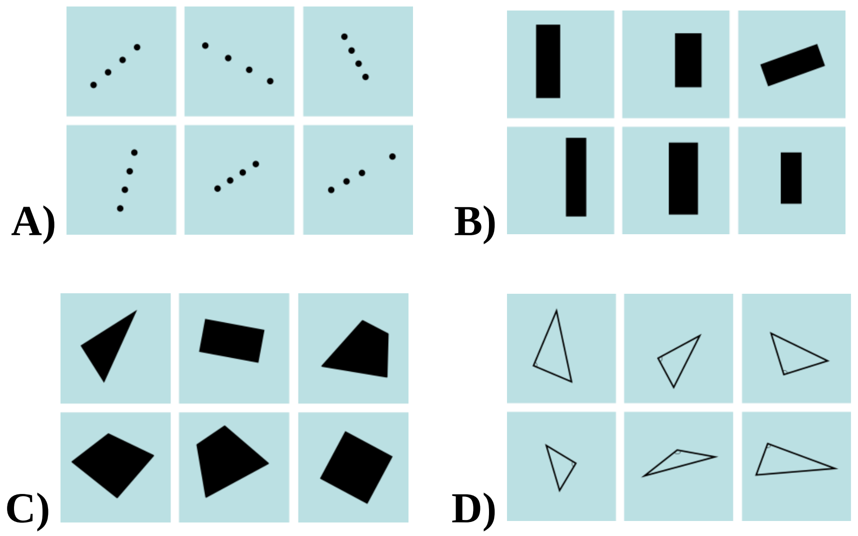


Figure 8.2 C and D can only be solved by considering edges.

However, Figures 8.2C and 2D can only be solved at the edge level. This is because there are no object-level attributes indicating that an object has four sides or that an object includes one right angle. Of course, one *could* encode such features at the object level. My claim is that these are not natural components of the object-level representation—they describe geometric features of the edges, rather than of the overall shape. In the Simulation section below, we examine whether people perform differently on problems like 2C and 2D.

The model first attempts to solve a problem at the group level. If it fails to identify a unique answer, then it reverts to the edge level. It does not also attempt the object level, as group-level and object-level representations are frequently the same.

Filtering out Information

In some cases, it is immediately clear that particular features are irrelevant in solving a problem. For example, in Figure 8.3A, the images all contain lines at different orientations. Therefore, orientation is unlikely to be a relevant feature, and so the fact that the upper right image contains axis-aligned edges is probably unimportant. In Figure 8.3B, the squares are all different sizes. Therefore, size is unlikely to be a relevant feature, and so the fact that the upper right image contains the largest square is probably unimportant. In Figure 8.3C, the objects are all different shapes--they are all triangles, but there are different angles between the triangles' edges. Because there is no simple transformation (rotation, rescaling, etc) between the shapes, it is difficult to compare sizes. Therefore, again, size is unlikely to be a relevant feature, and so the fact that the lower right triangles are the largest is probably unimportant.

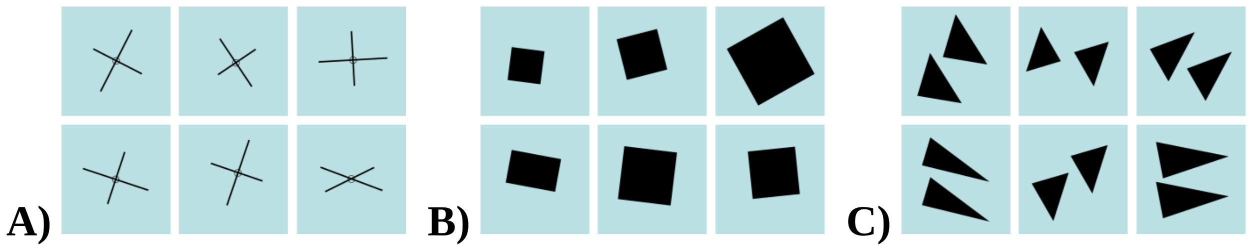


Figure 8.3 Certain information can be filtered out.

The model attempts to capture this process of noting that a feature type is irrelevant and filtering it out before problem-solving. It does this by calling *Find-Differences* on the entire array of six images, with the **:first-to-last?** parameter set to true. This means it creates a single list of six images and compares every adjacent pair, as well as comparing the last image (the lower right) to the first image (the upper left). It checks the results for non-adjacent differences of the relevant type (see the section on *Find-Differences* in section 4.2). For example, in Figure 8.3A, there are orientation differences between the first two images (upper left and upper middle) and between the last two images (lower middle and lower right). Because these image pairs are non-adjacent, the differences cannot be blamed on a single image, the odd the image out. The model can conclude that the images truly do vary in their orientations, so orientation may not be a relevant feature for problem-solving.

The model filters out orientation if the **:non-adjacent-orientation-differences?** result is true. *Find-Differences* will note orientation differences whenever there is a change to an orientation-specific feature (such as **rightOf** or **2D-Shape-Axis-Aligned**) or there is a rotation or reflection between two objects' shapes.

The model filters out size if the (**:non-adjacent-differences-of-types :size :shape**) result is true. This means there are non-adjacent differences in either size or shape. *Find-Differences* will note size differences when there is a change in a size attribute or when there is a scaling transformation between shapes. It will note a shape difference when corresponding objects are not the same shape (there is no valid transformation between them).

The model filters out a feature type by updating the *Perceive* operation to remove all features of the relevant type.

Similarity Measure

Usually, the odd image out lacks some feature found in the other images. For example, in Figure 8.4A, the upper right image lacks parallel edges. However, sometimes the odd image out possesses an extra feature. In Figure 8.4B, the lower middle image is the *only* image with parallel edges. In Figure 8.4C, the lower right image is the only image with rotated shapes.

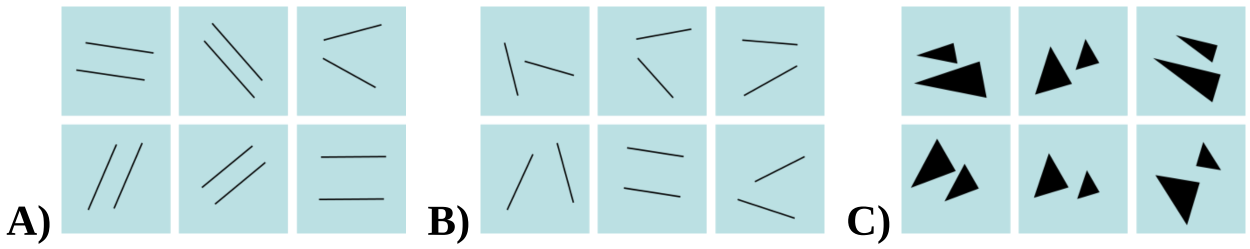


Figure 8.4 B and C require a different similarity measure.

The model uses different similarity measures, depending on whether it's expecting the odd image out to have a missing or extra feature. The first measure asks how well an image covers everything in the generalization. In Figure 8.4A, the upper right image doesn't cover the bottom-row generalization very well because it lacks the parallel edges. The second measure asks how well the image and the generalization cover each other. In Figure 8.4B, the lower middle image covers everything in the generalization. However, the generalization doesn't cover the image very well because it lacks the parallel edges.

The **:coverage** parameter determines how SME similarity scores are normalized (see section 4.2). With **:base** coverage, the score is normalized for the size of the generalization. If the image is missing features found in the generalization, it will receive a lower score. With **:symmetric** coverage, the score is normalized for the sizes of the generalization and the image combined. If either lacks a feature found in the other, the score will suffer.

The model uses **:base** coverage by default. That is, it first checks whether an image is missing some feature. If the model fails to find a unique answer, it reverts to **:symmetric** coverage.

Strategic Shifts

The model begins with the **:base** coverage similarity measure. It iterates over two levels of abstraction: groups and edges. At each level, it filters out irrelevant information and then attempts to solve the problem. If the model fails to find an answer at either the group or edge level, then it reverts to the **:symmetric** coverage similarity measure as a last resort.

8.3 Simulation

The simulation used the stimuli from (Dehaene et al., 2006) as input. The original experimenters provided PowerPoint slides with 41 of the 45 stimuli, and I recreated the other four before importing the 45 slides into CogSketch. One problem was touched up in PowerPoint: separate parts of a shape were redrawn as a single shape. Additionally, five were altered in CogSketch: extra lines meant to draw the participant's attention to a particular feature were removed, since CogSketch cannot use this information. Aside from these minor changes, all stimuli were identical to those used in the original study.

I compared the model's performance to participants from (Dehaene et al., 2006), considering only participants for whom complete data was available. The four groups considered were 40 young children (Americans, aged 4-8 years, mean = 6.12 years, 24 female/16 male), 64 older children (Americans, aged 8-12 years, mean = 10.40 years, 33 female/31 male), 47 adults (Americans, aged 18-52, mean = 26.41 years, 28 female/19 male), and 44 Mundurukú (aged 5-83 years, mean = 37.73 years, 24 female/20 male). I collapsed across all Mundurukú ages because Dehaene et al. found no age-related differences in Mundurukú performance.

I analyzed the model's results, asking the Sufficiency, Similarity, and Explanation questions.

Sufficiency

Overall, the model correctly solves 39/45 problems, or 87%. This places it above the average performance for the human groups (see Table 8.1), demonstrating that the model is sufficient for performing the task.

	Accuracy
Model	0.87
Young Children	0.55
Older Children	0.75
Adults	0.83
Mundurukú	0.67

Table 8.1 Accuracy of the model and each participant group on the 45 oddity task problems.

Similarity

Table 8.2 shows the correlations between the model's accuracy and each group's accuracy on the 45 problems, as well as correlations between groups—note that the model's accuracy on a given problem was always either 0% or 100%. The model correlates significantly with all groups ($p < .05$), suggesting that the problems on which the model failed were also difficult for human participants. The model has the highest correlation with American adults (r = .77), and the lowest correlation with the Mundurukú (r = .49).

	Model	Young Children	Older Children	Adults	Mundurukú
Model	1	.58	.66	.77	.49
Young Children	.58	1	.91	.84	.83
Older Children	.66	.91	1	.94	.75
Adults	.77	.84	.94	1	.72
Mundurukú	.49	.83	.75	.72	1

Table 8.2 Correlations in accuracy on each of the 45 problems (Pearson's r).

To explore the relationship between the model's performance and human error patterns, I ranked each problem by difficulty for a given group, such that 1 indicated the easiest problem for that group and 45 indicated the most difficult. I then considered each group's rankings for the six problems on which the model failed (see Table 8.3). For all groups, the median ranking is 41 or above. For all American groups, the minimum rank is above 30/45, meaning all six problems were difficult for those groups. For the Mundurukú, one problem ranks much lower (12/45). This problem (Figure 8.5) is unique in that it was difficult for both the model and the Americans, but relatively easy for the Mundurukú.

	Median Rank	Min Rank	Max Rank
Young Children	41.5	34	45
Older Children	42	33	45
Adults	42.5	37	45
Mundurukú	41	12	44

Table 8.3 Rankings of the 6 problems the model failed to solve (1 = easiest, 45 = hardest).

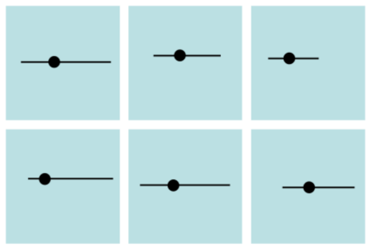


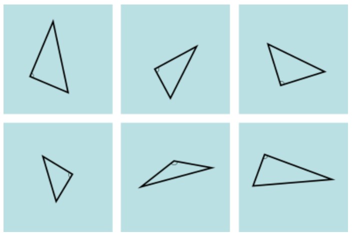
Figure 8.5 One of the six problems the model failed to solve. Average human performance: 68% (American adults), 86% (Mundurukú).

Explanation

The oddity task is arguably simpler than the previous tasks because it does not require reasoning about abstract sets of differences. In explaining human performance, I focused on spatial representation, rather than problem-solving strategies. In particular, I looked at abstraction. Firstly, problems might be more difficult for one group or another because they require reasoning at a certain level of abstraction. Thus, I ran simulations in which I ablated the group or edge levels.

For the Groups factor, I ablated the ability to build group-level representations. Instead, the model attempted each problem at the object level and at the edge level. Thus, it still considered two levels of abstraction, but it never grouped objects together.

For the Edges factor, I ablated the ability to build edge-level representations in some cases. Based on my previous simulations (Lovett & Forbus, 2011), I suspected that the edge level might be more difficult with closed shapes than with open shapes. That is, with closed shapes (e.g., Figure 8.6A), participants' attention might be drawn to the overall shape, and they might have trouble considering the individual edges, but with open shapes (e.g., Figure 8.6B), participants might more easily attend to individual edges. Thus, I selectively ablated the ability to build edge-level representations for closed shapes.

A) 

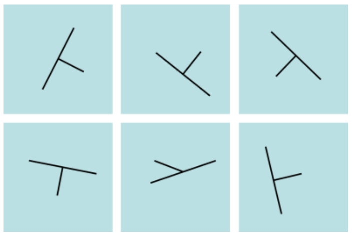
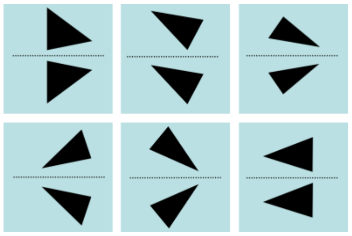
B) 

Figure 8.6 Problems with closed and open shapes.

I also considered relational abstraction. Some oddity task problems (e.g., Figure 8.7A) rely on the ability to compare objects and notice a transformation within each image. As discussed earlier, this may be difficult either because participants don't bother to compare objects or because they fail to compute the transformation. Other oddity task problems rely on the ability to notice symmetry in complex shapes (e.g., Figure 8.7B). According to the model, this requires a similar comparison; the shape is compared to itself to compute symmetry (Ferguson, 1994).

Thus, I produced two additional factors: Shape Transformation and Shape Symmetry. For Shape Transformation, I ablated the ability to compute transformation relations within an image. For Shape Symmetry, I ablated the ability to compute axes of symmetry. Note, however, that one form of symmetry was still available: the **Non-Elongated-Shape** attribute. This attribute (see section 3.2), used when other symmetry information is unavailable, describes a shape that lacks an axis of elongation. Thus, it can distinguish a square from a rectangle, or an equilateral triangle from an isosceles triangle. However, it cannot identify symmetry based on the angles between edges (as in Figure 8.7B).

A) 

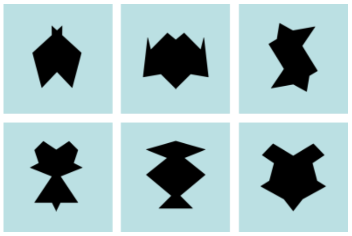
B) 

Figure 8.7 Problems relying on (A) shape transformation and (B) shape symmetry.

Accuracy

I built linear models for each group's accuracy, using the four factors as independent variables (Table 8.4). All four linear models have R^2 values of around .5, indicating they account for about half the variance in human performance. Importantly, there are differences in which factors contribute significantly to each group's model. In particular, the Edges factor contributes significantly in two of the American models (young children and older children), and is marginally significant ($p = .058$) in the third model (adults). In contrast, the Groups factor contributes significantly in the Mundurukú model. Thus, Americans appeared to have greater difficulty focusing on the individual edges of objects, rather than the objects as a whole (e.g., Figure 8.6A), whereas the Mundurukú had greater difficulty looking at overall configurations of objects (e.g., Figure 8.2A).

The other difference may be related: whereas all groups had greater difficulty on problems involving Shape Transformation, only the Americans had difficulty on problems involving Shape Symmetry. These problems (e.g., Figure 8.7B), appear to require a close consideration of the edges.

Recall that in the previous two models, working memory load was a factor when participants had to remember abstract sets of differences. In the present task, participants don't have to remember differences. Thus, one might expect working memory load to be less of a factor. I tested this by adding Elems2 to the linear models. Elems2 refers to the number of elements in the row generalization. Again, the first two elements aren't counted. Table 8.5 shows the results (compare to Table 8.4). As we see, Elems2 was not a significant contributor in any of the models. Interestingly, for the American participants, adding this factor either weakened or strengthened the Edges factor. This may be because problems typically have more edges than objects—thus, when a problem must be solved at the edge level, there are usually more elements. However, none of the models show a considerable improvement over the models without Elems2.

Model	R² (adj R²)	Intersect	Groups	Edges	Shape Transformation	Shape Symmetry
Young Children	**.57 (.52)**	0.71	-0.07	-0.21	-0.34	-0.41
Older Children	**.61 (.57)**	0.91	-0.04	-0.20	-0.28	-0.39
Adults	**.47 (.40)**	0.94	-0.01	-0.08*	-0.18	-0.17
Mundurukú	**.57 (.52)**	0.81	-0.31	-0.07	-0.42	-0.14

Table 8.4 Linear models for each group's accuracy. Grayed cells show significant contributors ($p < .05$). (* $p = .058$)

Model	R² (adj R²)	Intersect	Groups	Edges	Shape Transformation	Shape Symmetry	Elems2
Young Children	**.57 (.51)**	0.71	-0.08	-0.26	-0.35	-0.41	0.03
Older Children	**.63 (.57)**	0.91	-0.07	-0.31	-0.29	-0.39	0.07
Adults	**.50 (.42)**	0.94	-0.04	-0.20	-0.18	-0.16	0.07
Mundurukú	**.57 (.50)**	0.81	-0.32	-0.09	-0.42	-0.14	0.01

Table 8.5 Linear models for each group's accuracy, with Elems2. Grayed cells show significant contributors ($p < .05$).

Reaction Times

Finally, I built linear models for the average reaction times of each group. This analysis was restricted to the three American groups, as precise timing data was unavailable for the Mundurukú. It was also restricted to 37 of the 39 problems, since the first two were used in (Dehaene et al., 2006) as training problems. I considered only correct responses.

The results (Table 8.6) show a clear split between the young children and the other groups. The model for the young children, while a marginally significant predictor ($p = .052$), accounts for substantially less of the variance than the models for the older children and adults. This may be because the young children were less systematic or more easily distracted while performing the task.

The timing models for the older children and adults both account for over half the variance. Every factor is a significant contributor. This suggests that objects are the most basic level for American representations of two-dimensional space. Participants required more time on problems that involved working with a different level of representation (Groups or Edges). They also required more time on problems involving Shape Transformation or Shape Symmetry, confirming the model's prediction that these features require additional operations to compute.

Model	**R^2 (adj R^2)**	**Intersect**	**Groups**	**Edges**	**Shape Transformation**	**Symmetry**
Young Children	**.24 (.15)**	8.5	1.7	1.2	5.1	0.5
Older Children	**.69 (.65)**	5.9	3.0	5.0	4.9	5.1
Adults	**.63 (.58)**	5.4	2.9	3.6	5.4	4.1

Table 8.6 Linear models for each group's reaction times (in s). Grayed cells show significant contributors ($p < .05$).

Note that the reaction time advantage for objects over groups does not contradict the claim that groups are available first during reasoning. The object-level problems were not problems where individuals first encoded at the group level and then moved down to the object level. They were problems where there were no groups, and so the initial encoding described only objects. People reacted faster when the initial encoding described only objects and slower when the initial encoding described groups. This may indicate that the initial, pre-attentive processing required more time to build up to group-level representations. Note that the cost for groups was less than the cost for any of the other factors.

8.4 Discussion

The simulation suggests cultural differences in spatial cognition may manifest as differences in representational focus (Nisbett & Masudo, 2003; Nisbett & Miyamoto, 2005). Americans are biased to focus first on objects and object groups, leading to greater difficulty reasoning about the edges within an object. The Mundurukú, in contrast, do well with edges or objects, but have more difficulty considering entire groups of objects. Indeed, the impressive performance of the Mundurukú on one problem that was difficult for both the Americans and the model (Figure 8.5) suggests they may divide up space to consider even quadrants within a single edge.

One possible reason for this cultural difference is that formal schooling in geometry, and the many shape names in the English language, might focus Americans on object shapes when they consider visual scenes. The Mundurukú, lacking such exposure to formal geometric concepts, may be more inclined to focus on the parts that make up a shape. Understanding these cultural differences and how they arise, using a combination of psychological and computational studies, will help shed light on what is universal versus not in human spatial cognition.

I now consider two general factors that appeared to be difficult for both Americans and the Mundurukú.

Relational Abstraction

The constant across all groups was a cost for shape transformation. That is, all groups had trouble reasoning about shape transformations within an image. This suggests that relational abstraction is a major factor in the oddity task. Interestingly, there are also two problems involving shape transformations *between* images (Figure 8.8). The model fails to solve these problems. Recall that, in the model, finding differences relies on analytic shape comparison. That is, when the model is identifying differences between two images, it compares the corresponding shapes to identify shape transformations. However, generalizing does not. When the model generalizes over multiple images, it does not consider how shapes in separate images relate to each other. Thus, the model cannot solve the problems in Figure 8.8[9], and it predicts that these problems will be difficult for human participants.

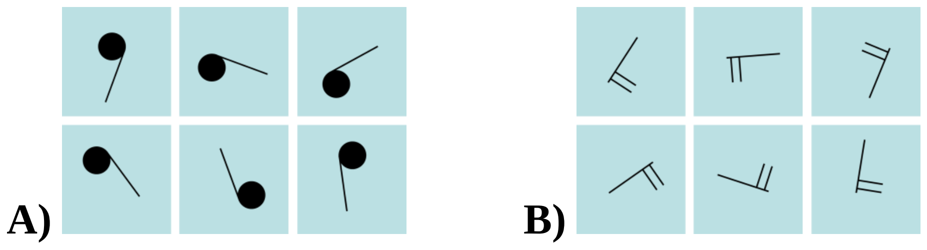


Figure 8.8 Problems requiring shape comparison between images.

In fact, these problems are quite difficult. For the older children, they are among the 12 hardest problems. For all other groups, they are among the eight hardest problems. They are particularly difficult for young children and the Mundurukú, who both average .23 accuracy on these problems. All groups perform worse on these problems than on the problems with shape transformations within each image.

Thus, relational abstraction is again a major source of difficulty. In the previous tasks, pairwise image comparisons may have cued participants to look for shape transformations—indeed, the geometric analogy problems with transformations were quite easy. However, generalizing over several images apparently does not. The problems in Figure 8.8 require significant effort.

Qualitative Representations

One of my core claims it that people use qualitative representations whenever possible. The oddity task model solves each problem by identifying a *qualitative* difference between one image and the others. Thus, the model predicts that problems will be more difficult when there is only a *quantitative* difference.

[9] Figure 8A presents an additional challenge: the model would not normally group a line and a circle together to form a single object. However, this limitation of the model is not important for the present discussion.

Figure 8.9 shows two other problems the model fails to solve. In each case, the odd image out lacks a quantitative feature. In Figure 8.9A, the objects are rotated a different amount. In Figure 8.9B, the dot is located at a different percentage of the edge's length. Are these problems difficult for humans?

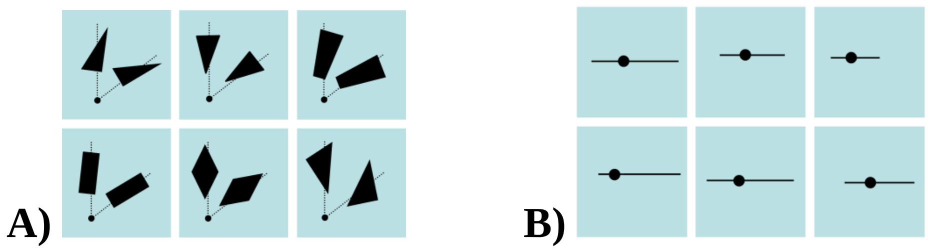


Figure 8.9 Problems involving quantitative features.

Figure 8.9A, which combines quantitative features and relational abstraction, is extremely difficult. It is among the four hardest problems for all groups. Figure 8.9B, which has been discussed before, is more interesting. It is relatively difficult for all the American groups: it is among the 13 hardest problems. This is particularly impressive given that it is such a visually simple problem, containing only an edge and a dot. However, the problem is far easier for the Mundrukú. They perform at .86 accuracy on this problem, well above their overall average of .67.

Rather than indicating that the Mundrukú are better at quantitative features, I believe this result suggests that the Mundrukú possess an additional *qualitative* feature. Given that they are better at breaking an object down into its edges, they may also be better at breaking an edge down into its quadrants. If so, they may go beyond the model, which qualitatively encodes when an object is centered on an edge. They may encode when an object is centered on one *half* of an edge. Of course, further study is needed to test this interpretation.

Does this mean people are universally bad at reasoning over quantitative information? Not necessarily. Consider Figure 8.10. Intuitively, the distance between the dot and the edge appears far greater in the lower right image. Note that the model solves this problem using qualitative information: the edge is centered in every image except the lower right, where it is on the lower half of the image. It is possible that this qualitative difference draws people's attention to the quantitative difference in distance. However, one cannot rule out the possibility that the quantitative information alone is sufficient for solving this problem. Further oddity problems that control for the qualitative difference would be needed to resolve this question.

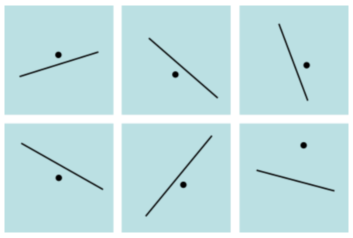


Figure 8.10 Problem in which the quantitative difference is particularly salient.

It is entirely possible that people *can* reason and generalize over quantitative information. I suspect that this is far easier for concrete information that is directly available in the image. People may have less access to second-order information which requires effort to compute, such as degrees of rotation or percentage of an edge's length (Figure 8.9). When people generalize over several images, they may include concrete quantitative features in their generalization.

Note that this does not contradict my original claim, that qualitative information is used first during comparison. Individuals might compare two images by aligning their qualitative, relational structure. For example, they would compare the first two images in Figure 8.10, aligning the dot with the dot and the edge with the edge. Afterwards, they might consider quantitative information, such as the distance or angle between two objects. If this quantitative information is constant, it might be included in the generalization.

In the future, I hope to further evaluate people's ability to reason and generalize over quantitative information. If people prove capable, I would like to add this ability to the modeling framework.

9. Predictions

Psychological theories rarely provide concrete descriptions of people's representations and processes. Thus, when a computational model is developed, the modeler must make several assumptions, based upon what is known about cognition and what is required by the target task. These assumptions, in turn, become predictions about how people may perform the task. Other predictions can emerge from the model's behavior, without any prior expectation by the modeler. In a symbolic computational model, such as the ones I have developed, these predictions are often explicit and clear. They can be tested in new psychological studies. In this way, we can refine both the model and our understanding of human cognition.

In this chapter, I enumerate several testable predictions arising from the models. The predictions are divided into two categories: visual comparison and spatial problem-solving.

9.1 Visual Comparison

1) Similarly shaped objects needn't always align.

Some computational models (e.g., Carpenter, Just, & Shell, 1990) simplify image comparison by assuming that similarly-shaped objects will always be placed into correspondence. In contrast, the Spatial Routines model does not require this. Similarly-shaped objects will share several shape attributes. Thus, they will usually be aligned. However, if there is sufficient pressure from the relational structure, differently shaped objects may be aligned instead.

For example, consider the two images in Figure 9.1A. These images contain little relational structure. Thus, it is easy to override this structure and map the circle to the circle and the triangle to the triangle. Contrast this with Figure 9.1B. Here, the images contain far more structure. The model predicts that in this case, people are more likely to map the circle to the triangle and the triangle to the circle.

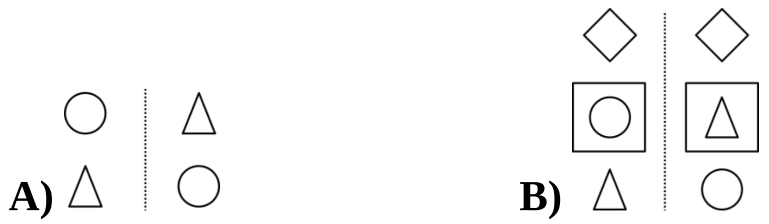


Figure 9.1 Two image pairs. It is harder to align the same-shaped objects in B than in A.

2) Shape comparison and symmetry detection use similar representations and processes.

When the model compares two objects' shapes, it first uses the Structure-Mapping Engine (SME) to compute a mapping between their edge-level representations. It computes a transformation based on one corresponding edge pair and then applies this transformation to one object's entire shape. The process for computing axial symmetry is similar. There are two primary differences: 1) For symmetry, the model uses MAGI (Ferguson, 1994), a variant of SME designed to produce self-mappings. 2) The representation for symmetry is slightly different; it does not order edges clockwise around the object's contour. Recall, however, my suggestion that people likely don't order edges clockwise even during shape comparison.

Thus, the model predicts parallels in shape comparison and symmetry detection. Whatever makes one difficult (e.g., shape complexity) should make the other difficult. That doesn't always mean that if someone is bad at one, they will be bad at the other. It is possible that someone will be more inclined to compute one than the other, e.g., the Mundurukú did better on oddity task symmetry problems than rotation/reflection problems (Dehaene et al., 2006).

3) Qualitative differences should be noticed faster than quantitative differences.

This prediction has already been borne out in several cases—see the discussion of categorical perception in Chapter 2. However, I wish to state it more generally. During image comparison, people should notice qualitative differences faster than quantitative differences. Furthermore, this effect should become stronger as images become more complex—I predict that when faced with complex stimuli, which place a greater load on working memory, participants will rely more on qualitative representations due to their abstract, concise nature.

For example, in Figure 9.2, the quantitative distance changes the same amount in A and B. However, in B there is an accompanying qualitative change: the circles now intersect. Figure 9.3 presents similar changes but in a more complex image. I predict that B will be easier to spot than A in both cases, but the difference will be greater in Figure 9.3.

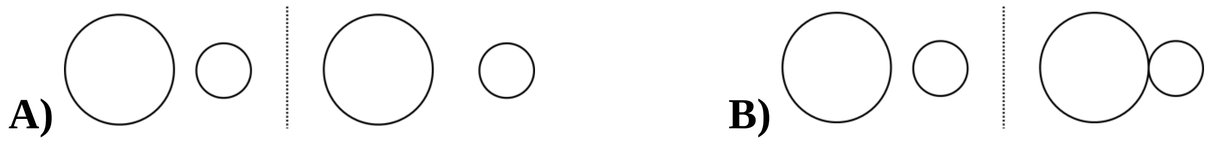


Figure 9.2 Can you spot the difference between each image pair?

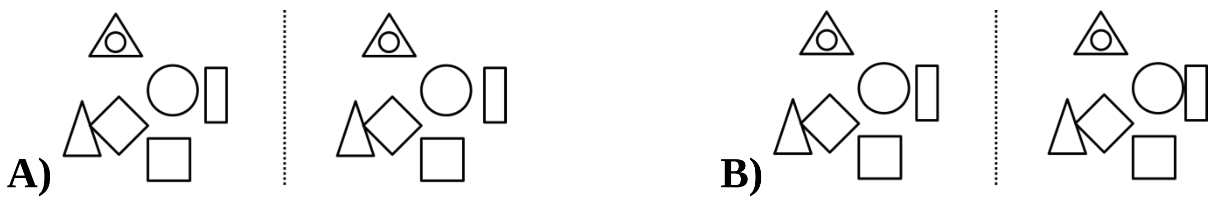


Figure 9.3 Can you spot the difference between the more complex image pairs?

Limited support for the more general prediction comes from a study by Stollinski et al. (2009). They showed participants pairs of complex line drawings (for example, drawings of an oven) and ask them whether the two drawings depicted the same category. Participants were more likely to respond "no" if there were differences in the drawings' qualitative topological relations than if the differences left those relations intact. However, the experimenters did not control for the degree of quantitative change in the two conditions, so one cannot be certain that the qualitative relations were the key.

If the model's prediction is correct, it provides some useful opportunities for further study. We should be able to determine which qualitative features people encode by noting which differences are easier to spot between images. There might well be individual differences in people's qualitative representations, or in the amount they encode before comparing. Thus, we can expect features to vary along a spectrum of salience.

4) Large-scale differences should be noticed faster than small-scale differences.

Again, we have already seen some examples of this: people may notice differences in large-scale groups before small-scale groups (Navon, 1977), or in small-scale groups before individual objects (Love, Rouder, and Wisniewski, 1999). However, I further predict that people will notice differences in objects before differences in an object's individual edges.

This effect depends on what is encoded at the object level. For example, I have proposed that when a feature holds for all an object's edges, that feature "bubbles up" to the object level. If this is true, individuals should notice when an object has all straight or all curved edges. There should be a qualitative, object-level change if we flip the curvedness of one of the object's edges (Figure 9.4A, 9.4B). In contrast, suppose an object has both curved and straight edges. Flipping the curvedness of one edge should not change the object-level representation, and so it should be less salient (Figure 9.4C).

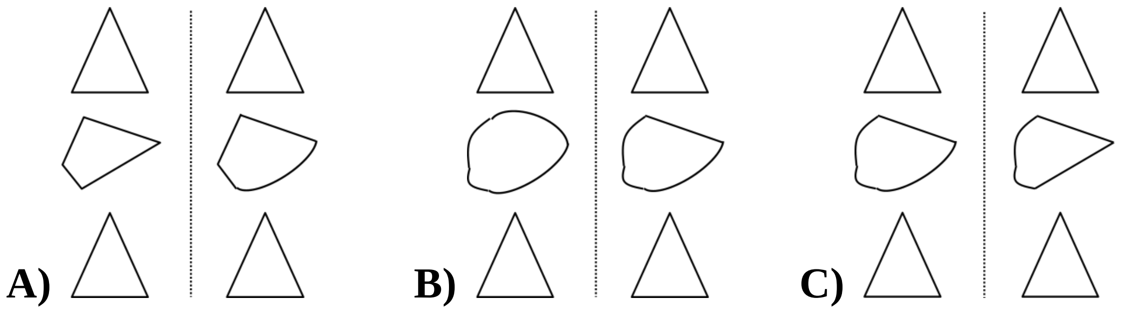


Figure 9.4 Object-level differences (A, B) may be more salient than edge-level differences (C).

Again, if this prediction is correct then we can use it to determine what people encode at the object or group level. We can evaluate how quickly people detect different types of changes to an object or group.

Note that predictions 3) and 4) align well with psychological research on change blindness. Change blindness (Levin & Simons, 1997; Rensink, O'Regan, & Clark, 1997) is a phenomenon where individuals fail to notice significant changes in a video or between two images. Unlike the other research I have described, change blindness studies often use detailed color photographs. In one of Rensink's famous examples, a display flickers continuously between two photos of soldiers boarding an airplane. It can take participants 10-20 seconds to notice that in one of the photos, the airplane is missing a large jet engine.

The model predicts when change blindness should occur. If a change does not affect someone's large-scale, qualitative representation, people are less likely to notice it. For example, a plane is equally recognizable with or without a jet engine. Thus, an individual's initial perceptual representation should be unaffected by the presence or absence of the engine. In contrast, a change that made the plane less recognizable might be notice more quickly. Similarly, a change in the type of plane, or even a change in the plane's qualitative color, might be noticed quickly.

9.2 Spatial Problem-Solving

5) Shape transformations should be more salient when looking for differences.

Spatial Routines for Sketches (SRS) uses separate operations when comparing to find differences and comparing to find commonalities. Comparing to find differences utilizes analytic image comparison; that is, after comparing two images, SRS compares their corresponding objects to seek out shape transformations. In contrast, comparing to find commonalities (generalizing) utilizes basic image comparison. Thus, the model predicts that when people are looking for differences, their attention will be guided to shape transformations. However, when people are looking for commonalities, they may not notice shape transformations.

This prediction has already been borne out in the problem-solving tasks. In the oddity task, which involves commonalities, participants had great difficulty noting that several images had rotations of the same shape (Figure 9.5A). However, in geometric analogy, which involves differences, participants had little trouble with shape rotations (Figure 9.5B).

Of course, these tasks involve different numbers of images. One might control for this by creating a 2x3 matrix such as Figure 9.6. This matrix could be either an oddity task problem (6A) or a Raven's matrix problem (6B). Thus, one could test whether shape transformations are more apparent when looking for differences, as in 6B. Unfortunately, there is a confounding factor: in the Raven's matrix problem, participants may actively rotate the shape to infer the missing image, rather than simply comparing shapes. Thus, testing this prediction could be difficult.

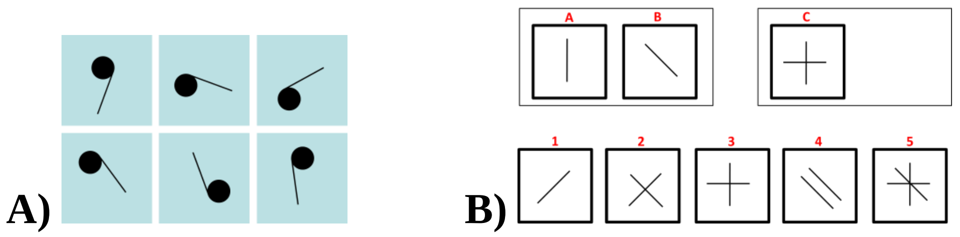


Figure 9.5 Problems involving shape rotations.

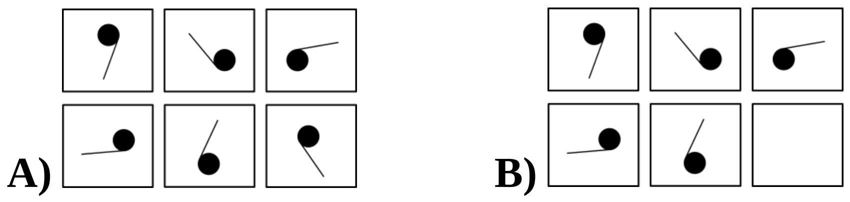


Figure 9.6 An oddity task problem (A) and a Raven's matrix problem (B) with identical images.

6) Visual inference shouldn't produce a concrete, quantitative representation.

The model uses visual inference to generate an image representation, for example, when solving for the answer to a geometric analogy problem. However, this process does not produce an actual image; it produces a qualitative image representation. Thus, the model predicts that people will not notice small, quantitative differences between an inferred image and an actual image.

Note that this prediction may result from a limitation of the current model. Recall that when the model performs *shape* comparison, it computes an exact, quantitative transformation, along with a qualitative transformation type. Therefore, when the model performs shape inference, it applies this transformation to the old shape to generate a concrete, quantitative representation of the new shape. Perhaps the model is limited because shape comparison produces quantitative information, while image comparison does not?

While I believe the model can and should consider more quantitative information at the image level (see the oddity task discussion in section 8.4), visual inference may still be an inexact process. Consider Figure 9.7, a variant of problem one from Evans (1968). Is the correct answer 2 or 5? Because the objects in A and C are different, it is difficult to apply precisely the same quantitative change in C as is found in A/B. All that is certain is that the "Z" shape should be to the left. Thus, while quantitative information may play a role in visual inference, I believe there is an inherent reliance on qualitative features.

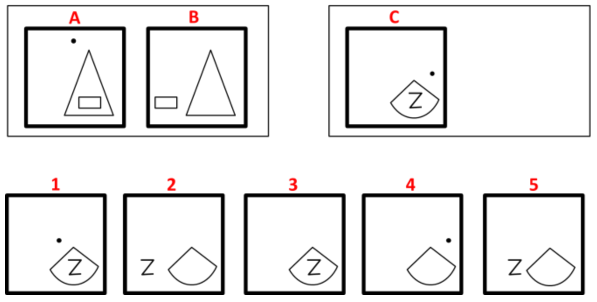


Figure 9.7 This geometric analogy problem requires quantitative information to solve.

7) During problem-solving, participants may select obvious shape transformations before correct shape transformations.

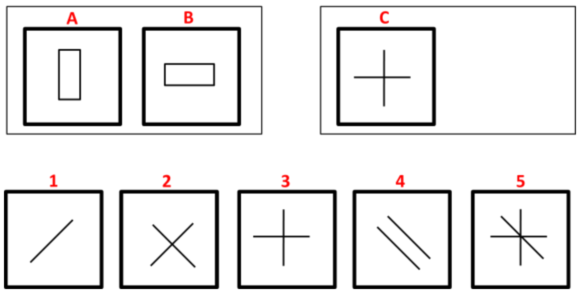


Figure 9.8 This geometric analogy problem presents a novel challenge.

Consider Figure 9.8, a variant of Evans (1968) problem two. Solving this problem via visual inference should be easy: you rotate the '+' sign 90° to get an identical '+' sign, so the answer is 3. However, according to the model, visual inference is difficult when transforming a shape produces the identical shape. Because the two '+' signs are identical, and because identity gets higher priority than rotation, there does not initially appear to be a 90° rotation between the '+' signs. Thus, the model treats the rotation as a failure, abandons visual inference, and reverts to second-order comparison.

At this point, the model makes a further prediction. In its first pass, it detects a rotation between C and 2, but not between C and 3 (again, C/3 appears to be identity). The C/2 rotation (45°) is not the same as the A/B rotation. However, minimal ascension allows non-identical rotations to align (see Chapter 4). The C/2 rotation is considered sufficiently close to the A/B rotation, so the model picks answer 2. If answer 2 weren't present, it would look for alternate shape transformations, and it would be eventually discover that answer 3 is better.

Thus, the model predicts that people will sometimes choose the wrong shape transformation when the correct shape transformation produces identical shapes. This prediction could be tested with problems like Figure 9.8. Of course, the prediction may be wrong. As I explain in the geometric analogy discussion (section 6.5), it may be that the A/B rotation primes people to look for rotations first, such that they immediately notice the rotation in C/3. I suspect individuals will vary in whether they choose the more obvious rotation (answer 2) or the more correct rotation (answer 3).

10. Conclusions

I have argued that human spatial problem-solving can be modeled as structural comparison and visual inference over qualitative, hierarchical representations. The previously described models meet three key criteria: 1) They perform as well as or better than typical human adults. 2) Problems that are hard for the models are hard for people. 3) The models can help explain what makes one problem harder than another. Each model has value on its own. However, by using identical representations and common processes across multiple task models, the current approach places more stringent constraints on the models and therefore strengthens them. Further work in other domains (e.g., psychology) will be required to test the models' predictions and evaluate how closely their representations align with our own.

Next, I return to the claims from the Introduction, considering how the thesis has addressed them. I then summarize related work on modeling problem-solving, analogical comparison, and qualitative representation. I close by describing avenues for future work.

10.1 Thesis Claims

The first five claims describe how spatial problem-solving can be modeled.

1) People typically use qualitative representations of space during problem-solving.

In the current approach, reasoning primarily happens at the qualitative level. This is the level at which images are compared and commonalities and differences are identified. The models consider quantitative information only when necessary, to compare shapes or to evaluate whether a qualitative difference is accurate.

As the models show, geometric analogy and Raven's matrices can be solved by identifying qualitative differences between images, while the oddity task can be solved by identifying qualitative commonalities. Furthermore, oddity task problems that depend on identifying *quantitative* commonalities are generally much more difficult. However, I believe there may be a role for basic quantitative features (distance, position) in this task—see section 10.3 below.

2) Spatial representations are hierarchical.

The current approach simplifies *hierarchical hybrid representations* down to three levels: edges, objects, and groups. The models show that moving flexibly between those groups is required for problem-solving. The oddity task model uses holistic top-down comparison (see section 2.5): the model begins at the group level, and if it fails to solve a problem, it moves down the hierarchy, representing each image at the edge level. The other models use analytic top-down comparison: they compare images at the group level and then, guided by that comparison, they compare corresponding groups at the object level and corresponding objects at the edge level. The models can also perform *perceptual reorganization*, changing the groupings at low levels so that objects will better align at high levels. Chapter 7 shows that perceptual organization is a significant predictor of problem difficulty on Raven's Progressive Matrices.

3) Spatial representations are compared via structure mapping.

Structure mapping plays a ubiquitous role in these models. At each level in the hierarchy, it can compare qualitative representations to compute similarity, identify corresponding elements, encode commonalities, or encode differences. Because structure mapping is a heuristic mapping process, it does not require an exhaustive search through all possible mappings. Thus, these models use a more general, more efficient mapping process than many existing problem-solving models.

4) Spatial transformations are computed using structure mapping and the spatial hierarchy.

The models perform shape comparison by structurally aligning at the qualitative level, spatially transforming at the quantitative level (based on one of the qualitative correspondences), and then comparing corresponding parts. This approach, motivated by research on mental rotation, can compute shape transformations for even complex objects. It supports rotation, reflection, changes in scale, various deformations, and shape symmetry (when combined with MAGI: Ferguson, 1994). Again, this approach demonstrates the ubiquity of structure mapping.

5) Spatial problem-solving requires control processes.

The Spatial Routines language includes control structures for evaluating a solution and backtracking if necessary. All three task models use these control structures to make strategic shifts during problem-solving. Furthermore, the analyses indicate that problems requiring strategic shifts are often more difficult for humans. Thus, effective strategic control is a key to advanced problem-solving.

The remaining claims involve sources of difficulty that can be studied using this modeling approach. Note that these sources are heavily interrelated. I briefly describe each below.

6) Encoding & abstraction

In many cases, a problem may be difficult because it requires entity abstraction or relational abstraction. Recall that entity abstraction involves moving between levels in the spatial hierarchy, while relational abstraction involves comparing two elements and noticing a relationship between them (e.g., a shape transformation). These factors can be studied by ablating the model's ability to *Perceive* at a particular level, or its ability to perform certain comparisons during problem-solving.

The analyses show that abstraction is a frequent source of difficulty. On the oddity task, North Americans had trouble moving down in the spatial hierarchy (towards edges), while the Mundurkú had trouble moving up (towards groups). Both had great difficulty with relational abstraction: encoding shape transformations and using them to reason about commonalities.

On Raven's Progressive Matrices, participants had difficulty abstracting away images and reasoning about each object in isolation. In addition, perceptual reorganization was a major source of difficulty. Perceptual reorganization (particularly complex reorganization) involves both entity and relational abstraction, as one must recognize that objects share common parts, break them down into those parts, and then group them back up based on the patterns of correspondences.

7) Working memory load

Much of the previous work on geometric analogy and Raven's Progressive Matrices showed that working memory was a factor. The current models contribute by providing an automated measure of each problem's working memory load. This can be computed by counting the number of elements in a given representation.

The analyses show that working memory load is particularly a factor when representing differences between images (i.e., non-literal patterns of variance). In both geometric analogy and Raven's Progressive Matrices, problems became harder as the number of elements in the differences increased. Importantly, this factor was non-linear. In geometric analogy, the linear model best matched human results when the first two elements were not counted; only elements *beyond* two contributed to difficulty. This may be because three or more elements was enough to overload participants' working memory capacities, requiring a strategy change (Mulholland, Pellegrino, & Glaser, 1980), or at least more frequent backtracking. This same working memory measure proved successful on the Raven's Progressive Matrices analysis.

It is telling that working memory load was a factor specifically when representing abstract *differences* between images. In geometric analogy, the number of objects in the differences was more important than the number of objects in the solution image. In Raven's Matrices, working memory was important for the *difference* patterns of variance, but not the *literal* patterns of variance. In the oddity task, which doesn't require reasoning over differences, working memory apparently was not a factor. Thus, these findings again underline the importance of abstraction ability. Abstract representations of differences appear to place more load on people's working memory, perhaps because we are less accustomed to using them.

8) Control processes

As mentioned above, all three tasks models used control processes to make strategic shifts. The analysis involved coding problems for which strategic shifts are required to solve them. In two of the models, this was done by ablating the ability to perform those shifts. In geometric analogy, it was done by simply identifying which shifts the model naturally made to solve a problem. This was done because geometric analogy was generally easier, and only reaction times were being modeled; thus, the question wasn't whether people could solve a problem, but how long it would take them.

Strategic shifts were a source of difficulty on both geometric analogy and Raven's Matrices. On geometric analogy, people were slower when the model had to switch to a second-order comparison strategy or find a different mapping between images. On Raven's Matrices, people were slower when the model had to switch to a simple pattern of variance or compare the first and last images.

Note that this factor is particularly difficult to isolate from the previous two. Firstly, many strategic shifts involve performing some type of abstraction. For example, the simple patterns of variance are more abstract. Similarly, switching between levels of abstraction on the oddity task could be seen as making a strategic shift.

Secondly, strategic shifts (and control processes in general) almost certainly place load on participants' working memory (Carpenter, Just, & Shell, 1990). Participants who are better able to remember every difference between objects may also be better at remembering which strategies have been tried and which strategies remain. Future analyses looking at individual differences may shed light on how much these factors overlap. For the present, I believe they should be studied separately, given their different manifestations in each problem.

10.2 Related Work

Analogy and analogical problem-solving are popular targets for modelers. Researchers in Hofstadter's Fluid Analogies Research Group (FARG) (Hofstadter, 1995) have built several computational models for what they term *microdomains*: small, highly specific analogical problem-solving domains. Included among these is Mitchell's (1993) Copycat, which solves letter string analogies, and French's (1995) Tabletop, which identifies corresponding elements in table settings. These models are characterized by tight interleaving of perceptual encoding and analogical mapping, such that what has been mapped previously is likely to affect what will be encoded in the future. This contrasts somewhat with the modeling approach described here, in which encoding and mapping are separate processes. Note, however, that encoding and mapping are highly interactive in the current approach. For example, comparison can guide perceptual reorganization.

While their models contain some important insights, FARG's focus on tightly constrained microdomains makes comparison against human performance difficult. The tasks they model have been studied little, if any, by psychologists. Furthermore, the tasks are often ambiguous, without a clear correct answer, making it difficult to objectively evaluate a model's performance. In comparisons between Copycat and human performance (Mitchell, 1993; Hofstandter, 1995), the answer most frequently chosen by Copycat was often not the answer most frequently chosen by people. Indeed, much of the group's analysis is focused on the models' abilities to work out particularly sophisticated or creative solutions, rather than solutions that align well with typical human performance.

Other models perform problem-solving by transferring solutions from analogous problems. Galatea (Davies & Goel, 2001) works specifically in the visual domain and requires the user to provide an analogy to a solved problem. EUREKA (Jones & Langley, 1995) is more domain-general and can retrieve analogous problems from memory.

Several general-purpose models of analogy have been constructed. Among these, the model most similar to SME is Keane and Bradshaw's (1988) IAM model. Unlike SME, which finds local matches in parallel, IAM incrementally adds pieces to a mapping. Several connectionist models have been developed, including ACME (Holyoak & Thagard, 1989); LISA (Hummel & Holyoak, 1997), DRAMA (Eliasmith & Thagard, 2001), and CAB (Larkey & Love, 2003). Due to limitations of the connectionist approach, it is unlikely that these models could handle large-scale comparisons between complex images or shapes. Additionally, these models have generally been used with hand-coded input, rather than representations generated automatically from a stimulus (but see Hummel & Biederman, 1992, for work on generating a neural network representation from a line drawing).

The problem of generating qualitative spatial descriptions was explored previously with Ferguson and Forbus's (1999) GeoRep. GeoRep automatically constructs qualitative descriptions of line drawings. These descriptions can be used as the input for tasks such as recognition and identifying symmetry. Veselova and Davis' (2004) system generates qualitative constraints describing a hand-drawn sketch. These constraints can be used to recognize other sketches of the same object. Museros and Escrig's (2004) system generates qualitative descriptions of polygons that can be compared to identify rotations between the polygons. Their approach differs from our own in that they use a specialized comparison algorithm designed specifically for identifying rotations.

Several AI and robotics researchers have developed systems based on Ullman's (1987) visual routines theory (e.g., Chapman, 1992; Horswill, 1995; Rao, 1998). These systems implement a set of basic operations that can be combined to perform some task, often visual comprehension. Unlike Spatial Routines, these systems do not implement analogical comparison, and they are generally not used to study human problem-solving.

10.3 Future Work

Here I consider several directions for future work. These can be divided into three categories: model improvements, new problem domains, and psychological research.

Model Improvements

This work has demonstrated several strengths of the current modeling approach. However, it has also demonstrated certain weaknesses in the perception and comparison processes. The model is also limited in its inability to learn new problem-solving strategies.

Perception. There are two ways that I would like to improve the model's perceptual abilities. Firstly, I would like to develop a *visual routines* language for computing qualitative spatial features. Recall that visual routines (Ullman, 1987) was the inspiration for Spatial Routines. According to visual routines, perceptual features can be computed by combing basic low-level operations, such as tracing along a curve or filling in a region. A visual routines model and a Spatial Routines model could constrain one another (Forbus, 2001): features computed by visual routines must be usable in spatial problem-solving, while features used in spatial problem-solving must be computable by visual routines. Thus this approach, if successful, would lend considerable strength to any claims about the spatial features people utilize during problem-solving.

I previously built a preliminary visual routines system capable of computing topological and positional relations (Lovett & Forbus, 2009a, 2009b). I would like to build a more robust model to support spatial problem-solving. Ideally, each visual routine would produce: a) a qualitative spatial feature, such as a **parallel** relationship between two edges; b) a quantitative confidence measure for the feature. This confidence measure would be usable during comparison (see below).

Secondly, I would like to develop another level of abstraction: three-dimensional objects. This is different from the current object level which primarily supports two-dimensional shapes. Recall that the model represents object shapes by tracing the edges along their external contour. This would be insufficient for representing three-dimensional object shapes.

Essentially, this requires two levels: a surface level and a full object level. Surfaces are two-dimensional edge cycles. Full objects are sets of connected surfaces. Fortunately, our research group has already made some progress on representing the surfaces that make up a three-dimensional object (McLure et al., 2011; Lovett, Dehghani, & Forbus, 2007). More work is required to enrich these representations and integrate them into Spatial Routines. In particular, the model must be able to compute shape transformations between three-dimensional shapes.

Modeling three-dimensional shape transformations will be particularly exciting because of the potential for modeling spatial visualization. Recall that spatial visualization is the ability to mentally manipulate spatial representations (McGee, 1979). Much of the work on this ability involves manipulating three-dimensional representations (e.g., Hegarty et al., 2007). If we can model this process, we can make important strides towards understanding spatial ability in people.

Comparison. The present model of image comparison relies almost exclusively on qualitative features. However, as discussed in Chapter 2, people utilize both qualitative and quantitative features during reasoning. There are two ways the comparison process might be changed to better handle quantitative information.

Firstly, quantitative information could be used to overcome the brittleness of qualitative representations. This brittleness arises because generating qualitative features necessarily relies on applying thresholds. For example, the difference in two lines' orientations must be below a threshold before they are considered **parallel**. Unfortunately, this means that two stimuli can be nearly identical, and yet they may produce different qualitative representations. Perhaps one pair of lines barely meets the threshold for **parallel**, while the other pair barely misses the threshold. The qualitative representations will indicate that they are different, when in fact they may be extremely similar.

One solution is to use quantitative information when evaluating differences. This means qualitative representations will still be used for the initial comparison. Afterwards, the model might iterate over each qualitative difference and evaluate it quantitatively. This could be done using a confidence value for each qualitative feature (see above). Suppose a qualitative feature is present in image A but absent in image B. This might be resolved in a couple ways: 1) The confidence in image A may be relatively low. The model could conclude, based on the comparison, that the feature is not present in either image. 2) The feature may be present in image B, but with a confidence level just below threshold. The model could conclude, based on the comparison, that the feature is present in both images. Thus, this process can both improve comparisons and lead to re-representation based on the comparisons (Medin, Goldstone, & Gentner, 1993).

Secondly, quantitative information could be used when representing generalizations. This possibility arose when discussing the oddity task model (section 8.4). Suppose the model compares two images and constructs a generalization from them. Again, the model would begin by comparing the qualitative representations. Afterwards, it might evaluate a small set of concrete quantitative features, such as the distance between two adjacent objects, or the angle between them. If these quantitative features are highly similar, they could be included in the generalization. Thus, a generalization might include the information that two objects are about an inch apart.

Quantitative information might be used similarly in other image comparisons. Suppose the model is computing similarity between two images. After comparing the qualitative representations, it might compare those concrete quantitative features. If there is a quantitative change, such as one object changing position relative to the others, this could be noticed even if it does not produce a qualitative change. Thus, the model would predict not that quantitative information is ignored, but that: a) it doesn't guide the comparison and correspondence-finding; and b) it will be noticed more slowly than qualitative information.

Learning. Finally, both visual routines and Spatial Routines would be more effective if new routines could be learned automatically. This is equivalent to learning a new strategy for solving a problem. It would allow the models to better scale up to the range of possible strategies. It might also allow the models to better explain performance on a task like Raven's Progressive Matrices, where the problems are designed to teach a particular strategy while becoming progressively harder.

New Problem Domains

If this modeling approach is truly general, then expanding to new problem domains should be relatively straightforward. Thus, new problem domains may be used to evaluate the generality of my thesis claims. Two problem domains closely related to the current set are the Advanced Progressive Matrices and Bongard problems (Bongard, 1970).

The Advanced Progressive Matrices (APM) is a more difficult version of the test modeled in Chapter 7. Thus, it is not truly a new domain. However, a new set of Raven's matrix problems may present novel challenges. As I suggest at the end of Chapter 7, I believe the greatest challenge from APM is to encoding: the task likely requires new encoding strategies for breaking down each image into the necessary components and representing the relevant features. This may explain why every computational model of APM has used hand-coded inputs, while some models of the Standard Progressive Matrices (SPM) have automatically encoded the inputs.

Bongard problems (Bongard, 1970) initially appear to be a much simpler task. An individual is shown two arrays of images and asked to identify the difference between the arrays. However, I believe Bongard problems will present the same challenges as APM. Indeed, Bongard problems were *designed* to be difficult for AI systems because of the challenges they present for encoding. Every Bongard problem is a search through encoding strategies. Once the images have been encoded correctly, the answer should be obvious, but identifying the correct encoding strategy can be quite difficult.

Given the complexity of these problems, are they worth modeling? That depends on the modeling approach. I believe an approach of hard-coding every possible encoding strategy would be unproductive. It might be sufficient for performing the tasks, but it would tell us little about human problem-solving, as humans presumably don't begin the task already knowing every strategy. Instead, these might be prime cases for modeling learning. A model that could learn new encoding strategies to perform these tasks would truly be approaching human-level intelligence in the problem-solving domain.

Psychological Research

A limitation of cognitive models is that they can never prove a person is reasoning in a particular way. Even if a model's behavior appears identical to humans, it is possible that two different internal systems produced that behavior. What a model can do is: a) demonstrate that a set of representations and processes is *sufficient* for performing some task; b) serve as a tool for analyzing each problem in a problem set (see the linear regressions in Chapters 6-8); and c) generate testable predictions based on interactions between the model and the stimuli.

Thus, a cognitive model in isolation is of little use. A cognitive model combined with experimental research can push the frontiers of cognitive science. In section 10.1, I acknowledged lingering questions about how abstraction ability, working memory capacity, and strategic control interrelate. These questions could be answered by designing new problem sets, in which these factors are varied independently, and studying individual differences in performance. In Chapter 9, I listed seven predictions of the models. Each prediction could be tested in psychological experiments. The results of these experiments would then inform revisions of the model. The ultimate goal is a cycle of modeling and testing that incrementally refines our understanding of human cognition.

References

Abravanel, E. (1977). The figural simplicity of parallel lines. *Child Development, 48*(2), 708-710.

Appelle, S. (1972). Perception and discrimination as a function of stimulus orientation: The “Oblique Effect” in man and animal. *Psychological Bulletin, 78,* 266-278.

Arce-Ferrer, A. J., & Guzmán E. M. (2009). Studying the equivalence of computer-delivered and paper-based administrations of the Raven Standard Progressive Matrices Test. *Educational and Psychological Measurement, 69,* 855-867.

Bethell-Fox, C. E., Lohman, D. F., & Snow, R. E. (1984). Adaptive reasoning: Componential and eye movement analysis of geometric analogy performance. *Intelligence, 8*, 205-238.

Bhatt, R., Hayden, A., Reed, A., Bertin, E., & Joseph, J. (2006). Infants' perception of information along object boundaries: Concavities versus convexities. *Experimental Child Psychology, 94,* 91-113.

Biederman, I. (1987). Recognition-by-components: A theory of human image understanding. *Psychological Review, 94,* 115-147.

Biederman, I., & Bar, M. (1999). One-shot viewpoint invariance in matching novel objects. *Vision Research, 39*, 2885-2899.

Biederman, I., & Bar, M. (1998). Same-different matching of depth-rotated objects. *Investigative Ophthalmology and Visual Science, 39,* 1113.

Biederman, I., & Gerhardstein, P. C. (1993). Recognizing depth-rotated objects: Evidence and conditions for 3D viewpoint invariance. *Journal of Experimental Psychology: Human Perception and Performance, 19*, 1162-1182.

Bongard, M. M. (1970). *Pattern Recognition*. Rochelle Park, N.J.: Hayden Book Co., Spartan Books.

Bornstein, M. H., & Korda, N. O. (1984). Discrimination and matching within and between hues measured by reaction times: Some implications for categorical perception and levels of information processing. *Psychological Research, 46*, 207-222.

Burke, H. R., & Bingham, W. C. (1969). Raven’s Progressive Matrices: More on construct validity. *Journal of Psychology: Interdisciplinary and Applied, 72*(2), 247-251.

Carpenter, P. A., Just, M. A., & Shell, P. (1990). What one intelligence test measures: A theoretical account of the processing in the Raven Progressive Matrices test. *Psychological Review, 97*(3), 404-431.

Cattell, R. B. (1963). Theory of fluid and crystallized intelligence: A critical experiment. *Journal of Educational Psychology, 54,* 1-22.

Chen, S., & Levi, D. M. (1996). Angle judgment: Is the whole the sum of its part? *Vision Research, 36*(12), 1721-1735.

Cirillo, S., & Ström, V. (2010). *An anthropomorphic solver for Raven's Progressive Matrices* (No. 2010:096). Goteborg, Sweden: Chalmers University of Technology.

Chapman, D. (1992). Intermediate vision: Architecture, implementation, and use. *Cognitive Science, 16*, 491-537.

Cohen, A.R., Stotland, E., & Wolfe, D.M. (1955). An experimental investigation of need for cognition. *Journal of Abnormal and Social Psychology, 51*(2), 291-294.

Cooper, L. A., & Shepard, R. N. (1973). Chronometric studies of the rotation of mental images. In W. G. Chase (Ed.), *Visual Information Processing*. New York: Academic Press.

Corballis, M. C., & Beale, I. L. (1976). *The psychology of left and right.* Hillsdale, N.J: Erlbaum.

Corballis, M.C., & Roldan, C.E. (1975). Detection of symmetry as a function of angular orientation. *Journal of Experimental Psychology: Human Perception and Performance, 1*, 221-230.

Davies, J., & Goel, A. (2001). Visual analogy in problem solving. *Proceedings of the International Joint Conference on Artificial Intelligence*.

Dehaene, S., Izard, V., Pica, P., & Spelke, E. (2006). Core knowledge of geometry in an Amazonian indigene group. *Science, 311*, 381-384.

Embretson, S. E. (1998). A cognitive design system approach to generating valid tests: Application to abstract reasoning. *Psychological Methods, 3*(3), 380-396.

Elder, J., & Zucker, S. (1993). The effect of contour closure on the rapid discrimination of two-dimensional shapes. *Vision Research 33*(7), 981-991.

Eliasmith, C., & Thagard, P. (2001). Integrating structure and meaning: A distributed model of analogical mapping. *Cognitive Science, 25*(2), 245–286.

Evans, T. (1968). A program for the solution of geometric-analogy intelligence test questions. In M. Minsky (Ed.), *Semantic Information Processing*. Cambridge, MA: MIT Press.

Falkenhainer, B. (1990). Analogical interpretation in context. *Proceedings of the 12th Annual Meeting of the Cognitive Science Society.* Cambridge, MA.

Falkenhainer, B., Forbus, K., & Gentner, D. (1989). The structure mapping engine: Algorithm and examples. *Artificial Intelligence, 41*, 1-63.

Farell, B. (1985). "Same"-"different" judgments: A review of current controversies in perceptual comparisons. *Psychological Bulletin, 98*(3), 419-456.

Ferguson, R. W. (1994). MAGI: Analogy-based encoding using regularity and symmetry. *Proceedings of the 16th Annual Meeting of the Cognitive Science Society*.

Ferguson, R. W., Aminoff, A., & Gentner, D. (1996). Modeling qualitative differences in symmetry judgments. *Proceedings of the 18th Annual Meeting of the Cognitive Science Society*.

Field, D. J., Hayes, A., & Hess, R. F. (1993). Contour integration by the human visual system: Evidence for a local "association field." *Vision Research, 33*(2), 173-193.

Forbus, K. (2001). Exploring analogy in the large. In D. Gentner, K. Holyoak, & B. Kokinov (Eds.), *Analogy: Perspectives from Cognitive Science*. Cambridge, MA: MIT Press.

Forbus, K., Gentner, D., Markman, A. B., & Ferguson, R. W. (1998). Analogy just looks like high level perception: Why a domain-general approach to analogical mapping is right. *Journal of Experimental & Theoretical Artificial Intelligence, 10*, 231-257.

Forbus, K., Nielsen, P., & Faltings, B. (1991). Qualitative spatial reasoning: The CLOCK project. *Artificial Intelligence, 51*(1-3), 417-471.

Forbus, K., & Oblinger, D. (1990). Making SME greedy and pragmatic. *Proceedings of the 12th Annual Meeting of the Cognitive Science Society.*

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, 3*(4). 648-666.

French, R. M. (1995). *The Subtlety of Sameness: A Theory and Computer Model of Analogy-Making*. Cambridge, MA: MIT Press.

Geiser, C., Lehmann, W., & Eid, M. (2006). Separating "rotators" from "nonrotators" in the mental rotations test: A multigroup latent class analysis. *Multivariate Behavioral Research, 41*(3), 261−293.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science, 7*, 155-170.

Gentner, D. (1989). Mechanisms of analogical learning. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning* (pp. 199-241). London: Cambridge University Press.

Gentner, D., Rattermann, M. J., & Forbus, K. (1993). The roles of similarity in transfer: Separating retrievability from inferential soundness. *Cognitive Psychology, 25*, 524-575.

Gilbert, A. L., Regier, T., Kay, P., & Ivry, R. B. (2006). Whorf is supported in the right visual field but not the left. *Proceedings of the National Academy of Sciences, 103*, 489–494.

Goldstone, R. L., & Medin, D. L. (1994). Time course of comparison. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 20*, 29-50.

Grudin, J. (1980). Processes in verbal analogy solution. *Journal of Experimental Psychology: Human Perception and Performance*, *6*(1), 67-74.

Gust, H., Kühnberger, K.U., Schmid, U. Metaphors and heuristic-driven theory projection (hdtp). (2006). *Theoretical Computer Science, 354*(1), 98–117.

Harris, L. J. (1972). Discrimination of left and right, and development of the logic of relations. *Merrill-Palmer Quarterly of Behavior & Development, 18*, 307-320.

Haun, D. B. M., Rapold, C. J., Call, J., Janzen, G., & Levinson, S. C. (2006). Cognitive cladistics and cultural override in Hominid spatial cognition. *PNAS, 103*, 17568-17573.

Hedges, L. V., & Chung, V. (in preparation). Does spatial skill predict who will become a scientist? An examination using data from two longitudinal studies.

Hegarty, M., Keehner, M., Cohen, C., Montello, D. R., & Lippa, Y. (2007). The role of spatial cognition in medicine: Applications for selecting and training professionals. In G. Allen (Ed.), *Applied Spatial Cognition*. Mahwah, NJ: Lawrence Erlbaum Associates.

Hermer-Vazquez, L., Moffet, A., & Mukholm, P. (2001). Language, space, and the development of cognitive flexibility in humans: The case of two spatial memory tasks. *Cognition, 79*, 263-299.

Hochstein, S., & Ahissar, M. (2002). View from the top: Hierarchies and reverse hierarchies in the visual system. *Neuron, 36*, 791-804.

Hoffman, D. D., & Richards, W. A. (1984). Parts of recognition. *Cognition, 8*(1–3), 65–96.

Hofstadter, D. (1995). *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. New York: Basic Books.

Holyoak, K. J., & Thagard, P. (1989). Analogical mapping by constraint satisfaction. *Cognitive Science, 13*(3), 295–355.

Horswill, I. (1995). Visual routines and visual search: A real-time implementation and an automata-theoretical analysis. *Proceedings of the International Joint Conference on Artificial Intelligence.*

Hummel, J. E., & Biederman, I. (1992). Dynamic binding in a neural network for shape recognition. *Psychological Review, 99*, 480-517.

Hummel, J. E., & Holyoak, K. J. (1997). Distributed representations of structure: A theory of analogical access and mapping. *Psychological Review, 104*, 427–466.

Hunt, E. (1974). Quote the raven? Nevermore! In L. Gregg (Ed.), *Knowledge and Cognition* (pp. 129–158). Hillsdale, NJ: Erlbaum.

Huttenlocher, J., Hedges, L. V., & Duncan, S. (1991). Categories and particulars: Prototype effects in estimating spatial location. *Psychological Review, 98*(3), 352-376.

Intraub, H. (1999). Understanding and remembering briefly glimpsed pictures: Implications for visual scanning and memory. In V. Coltheart (Ed.), *Fleeting Memories* (pp. 47-70). Cambridge, MA: MIT Press.

Jager, G., & Postma, A. (2003). On the hemispheric specialization for categorical and coordinate spatial relations: A review of current evidence. *Neuropsychologia, 41*, 504-515.

Janssen, A. B., & Geiser, C. (2010). On the relationship between solution strategies in two mental rotation tasks. *Learning and Individual Differences, 20*(5), 473-478.

Jones, R., & Langley, P. (1995). Retrieval and learning in analogical problem solving. *Proceedings of the 17th Annual Conference of the Cognitive Science Society*. Pittsburgh, PA.

Julesz, B. (1984). A brief outline of the texton theory of human vision. *Trends in Neurosciences, 7*(2), 41-45.

Just, S. B. (1979). *Spatial reasoning ability as related to achievement in a dental school curriculum.* Unpublished doctoral dissertation, Rutgers University.

Keane, M. T., & Bradshaw, M. (1988). The incremental analogy machine: A computational model of analogy.

In D. Sleeman (Ed.), *Third European Working Session on Machine Learning* (pp. 53–62). London: Pitman; San Mateo, CA: Morgan Kaufmann.

Kellman, P. J. (2000). An update on Gestalt Psychology. In B. Landau, J. Sabini, J. Jonides, E. L. Newport (Eds.), *Perception, Cognition and Language: Essays in Honor of Henry and Lila Gelitman* (pp. 157-190). Cambridge, MA: MIT Press.

Kosslyn, S. M. (2005). Mental images and the brain. *Cognitive Neuropsychology, 22*(3), 333-347.

Kosslyn, S. M. (1980). *Imagery and Mind*. Cambridge, MA: Harvard University Press.

Kosslyn, S. M., Koenig, O., Barett, A., Cave, C. B., Tang, J., & Gabrieli, J. D. E. (1989). Evidence for two types of spatial representations: hemispheric specialization for categorical and coordinate relations. *Journal of Experimental Psychology: Human Perception and Performance, 15*, 723-735.

Kosslyn, S. M., Murphy, G. L., Bemesderfer, M. E., & Feinstein, K. J. (1977). Category and continuum in mental comparisons. *Journal of Experimental Psychology: General, 106*(4), 341-37.

Kuehne, S., Forbus, K., Gentner, D., & Quinn, B. (2000). SEQL: Category learning as progressive abstraction using structure mapping. *Proceedings of the 22nd Annual Conference of the Cognitive Science Society.*

Kunda, M., McGreggor, K., & Goel, A. (2010). Taking a look (literally!) at the Raven's intelligence test: Two visual solution strategies. *Proceedings of the 32nd Annual Conference of the Cognitive Science Society.*

Laeng, B., Peters, M., & McCabe, B. (1998). Memory for locations within regions: Spatial biases and visual hemifield differences. *Memory & Cognition, 26*(1), 97-107.

Lamb, M. R., & Robertson, L. C. (1988). The processing of hierarchical stimuli: Effects of retinal locus, locational uncertainty, and stimulus identity. *Perception and Psychophysics, 44,* 172–181.

Larkey, L., & Love, B. (2003). CAB: Connectionist analogy builder. *Cognitive Science,* 27, 781-794.

Levin, D. T., & Simons, D. J. (1997). Failure to detect changes to attended objects in motion pictures. *Psychonomic Bulletin & Review, 4*(4), 501-506.

Levinson, S. C. (1996). Frames of reference and Molyneux's question: Cross-linguistic evidence. In P. Bloom, M. Peterson, L. Nadel, & M. Garrett (Eds.), *Language and space* (pp. 109-169). Cambridge, MA: MIT press.

Loewenstein, J., & Gentner, D. (1995). Relational language and the development of relational mapping. *Cognitive Psychology, 50*, 315-353.

Love, B. C., Rouder, J. N., & Wisniewski, E. J. (1999). A structural account of global and local processing. *Cognitive Psychology, 38*, 291-316.

Lovett, A., Dehghani, M., & Forbus, K. (2007). Incremental learning of perceptual categories for open-domain sketch recognition. *Proceedings of the International Joint Conference on Artificial Intelligence*. Hyderabad, India.

Lovett, A., & Forbus, K. (2011). Cultural commonalities and differences in spatial problem-solving: A computational analysis. *Cognition, 121*(2), 281-287.

Lovett, A., & Forbus, K. (2009a). Using a visual routine to model the computational of positional relations. *Proceedings of the 31st Annual Conference of the Cognitive Science Society*. Amsterdam, The Netherlands.

Lovett, A., & Forbus, K. (2009b). Computing human-like qualitative topological relations via visual routines. *Proceedings of the 23rd International Qualitative Reasoning Workshop.* Ljubljana, Slovenia.

Lovett, A., Gentner, D., Forbus, K., & Sagi, E. (2009b). Using analogical mapping to simulate time-course phenomena in perceptual similarity. *Cognitive Systems Research 10*(3): Special Issue on Analogies - Integrating Cognitive Abilities, 216-228.

Lovett, A., Tomai, E., Forbus, K., & Usher, J. (2009b). Solving geometric analogy problems through two-stage analogical mapping. *Cognitive Science 33*(7), 1192-1231.

Lowe, D. G. (1989). Organization of smooth image curves at multiple scales. *International Journal of Computer Vision, 3*, 119–130.

Lynn, R., Allik, J., & Irwing, P. (2004). Sex differences on three factors identified in Raven's Standard Progressive Matrices. *Intelligence, 32*, 411-424.

Maki, R. H. (1982). Why do categorization effects occur in comparative judgment tasks? *Memory & Cognition, 10*(3), 252-264.

Maki, R. H., & Marek, M. N. (1997). Egocentic spatial framework effects from single and multiple points of view. *Memory & Cognition*, *25*(5), 677-690.

Markman, A. B., & Gentner, D. (1996). Commonalities and differences in similarity comparisons. *Memory & Cognition*, 24(2), 235-249.

Marr, D. (1976). Early processing of visual information. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences, 275*(942), pp. 483-519.

Marr, D., & Nishihara, H. K. (1978). Representation and recognition of the spatial organization of three-dimensional shapes. *Philosophical Transactions of the Royal Society of London, Series B, Biological Science, 200*(1140), 269-294.

McGee, M. G. (1979). Human spatial abilities: Psychometric studies and environmental, genetic, hormonal, and neurological influences. *Psychological Bulletin, 86*(5), 889-918.

McLure, M. D., Friedman, S. E., Lovett, A., & Forbus, K. D. (2011). Edge-cycles: A qualitative sketch representation to support recognition. *Proceedings of the 25th International Workshop on Qualitative Reasoning*.

Medin, D. L., Goldstone, R. L., & Gentner, D. (1993). Respects for similarity. *Psychology Review, 100*(2), 254-178.

Mitchell, M. (1993). *Analogy-Making as Perception: A Computer Model.* Cambridge, MA: MIT Press.

Mulholland, T. M., Pellegrino, J. W., & Glaser, R. (1980). Components of geometric analogy solution. *Cognitive Psychology, 12*, 252-284.

Navon, D. (1977). Forest before trees: The precedence of global features in visual perception. *Cognitive Psychology*, *9*(3), 353–383.

Newcombe, N. C., & Uttal, D. H. (2006). Whorf versus Socrates, Round 10. *Trends in Cognitive Science, 10*(9), 394-396.

Nisbett, R. E., & Miyamoto, Y. (2005). The influence of culture: Holistic versus analytic perception. *Trends in Cognitive Science, 9*(10), 467-473.

Nisbett, R. E., & Masuda, T. (2003). Culture and point of view. *Proceedings of the National Academy of Sciences, 100*(19): 11163-11170.

O’Donoghue, D. P., Bohan, A., & Keane, M. T. (2006). Seeing things - Inventive reasoning with geometric analogies and topographic maps. *New Generation Computing*, *24*(3), 267-288.

Oppenheim, A. V., & Schafer, R. W. (2009). *Discrete-Time Signal Processing, Third Edition*. Upper Saddle River, NJ: Prentice Hall.

Özgen, E. (2004). Language, learning, and color perception. *Current Directions in Psychological Science, 13*, 95-98.

Palmer, S. E. (1977). Hierarchical structure in perceptual representation. *Cognitive Psychology, 9*, 441-474.

Palmer, S. E. (1980). What makes triangles point: Local and global effects in configurations of ambiguous triangles. *Cognitive Psychology, 12,* 285-305.

Palmer, S. E., & Hemenway, K. (1978). Orientation and symmetry: Effects of multiple, rotational, and near symmetries. *Journal of Experimental Psychology: Human Perception & Performance, 4*, 691-702.

Palmer, S., & Rock, I. (1994). Rethinking perceptual organization: The role of uniform connectedness. *Psychonomic Bulletin & Review, 1*(1), 29-55.

Pylyshyn, Z.W. (2002). Mental imagery: In search of a theory. *Behavioral and Brain Sciences, 25*(2), 157–182.

Pylyshyn, Z. W. (2001). Visual indices, preconceptual objects, and situated vision. *Cognition, 80*(1-2), 127-158.

Pylyshyn, Z.W. (1981). The imagery debate: Analogue media versus tacit knowledge. *Psychological Review, 88*(1), 16–45.

Primi, R. (2001). Complexity of geometric inductive reasoning tasks: Contribution to the understanding of fluid intelligence. *Intelligence, 30,* 41-70.

Ragni, M., Knauff, M., & Nebel, B. (2005). A computational model for spatial reasoning with mental models. *Proceedings of the 27th Annual Conference of the Cognitive Science Society.*

Ragni, M., Schleipen, S., & Steffenhagen, F. (2007). Solving proportional analogies: A computational model. Workshop on Analogies: Integrating Multiple Cognitive Abilities, CogSci '07.

Rao, S. (1998). Visual routines and attention. Unpublished doctoral dissertation, Massachusetts Institute of Technology, Cambridge, MA.

Rasmussen, D., & Eliasmith, C. (2011). A neural model of rule generation in inductive reasoning. *Topics in Cognitive Science, 3*(1), 140-153.

Raven, J., Raven, J. C., & Court, J. H. (2000a). *Manual for Raven's Progressive Matrices and Vocabulary Scales: Research Supplement 3. Neuropsychological Applications.* Oxford: Oxford Psychologists Press.

Raven, J., Raven, J. C., & Court, J. H. (2000b). *Manual for Raven's Progressive Matrices and Vocabulary Scales: Section 3.Standard Progressive Matrices.* Oxford: Oxford Psychologists Press.

Raven, J., Raven, J. C., & Court, J. H. (1998). *Manual for Raven's Progressive Matrices and Vocabulary Scales: Section 1. General Overview.* Oxford: Oxford Psychologists Press.

Regier, T., & Kay, P. (2009). Language, though, and color: Whorf was half right. *Trends in Cognitive Science, 13*(10), 439-446.

Rensink, R. A., O'Regan, J. K., & Clark, J. J. (1997). To see or not to see: The need for attention to perceive changes in scenes. *Psychological Science, 8*(5), 368-373.

Roberson, D., Davidoff, J., Davies, I. R. L., & Shapiro, L. R. (2005). Color categories: Evidence for the cultural relativity hypothesis. *Cognitive Psychology, 50,* 378-411.

Roberson, D., Pak, H., & Hanley, J. R. (2008). Categorical perception of colour in the left and right visual field is verbally mediated: Evidence from Korean. *Cognition, 107,* 752-762.

Rosielle, L. J., & Cooper, E. E. (2001). Categorical perception of relative orientation in visual object recognition. *Memory & Cognition, 29*(1), 68-82.

Sagi, E., Gentner, D., & Lovett, A. (in press). What difference reveals about similarity. *Cognitive Science*.

Schwering, A., Gust, H., Kühnberger, K., & Krumnack, U. (2009). Solving geometric proportional analogies with the analogy model HDTP. *Proceedings of the 31st Annual Conference of the Cognitive Science Society.*

Shea, D.L., Lubinski, D., & Benbow, C.P. (2001). Importance of assessing spatial ability in intellectually talented young adolescents: A 20-year longitudinal study. *Journal of Educational Psychology, 93*, 604–614.

Shepard, R. N., & Cooper, L. A. (1982). *Mental images and their transformations*. Cambridge: MIT Press.

Shepard, R. N., & Metzler, J. (1971). Mental rotation of three-dimensional objects. *Science, 171*, 701-703.

Sholl, M. J., & Egeth, H. E. (1981). Right-left confusion in the adult: A verbal labeling effect. *Memory & Cognition, 9*(4), 339-350.

Snow, R.E., & Lohman, D.F. (1989). Implications of cognitive psychology for educational measurement. In R. Linn (Ed.), *Educational Measurement, 3rd ed.* (pp. 263–331). New York, NY: Macmillan.

Snow, R. E., Kyllonen, P. C., & Marshalek, B. (1984). The topography of learning and ability correlations. In R. J. Sternberg (Ed.), *Advances in the Psychology of Human Intelligence* (vol. 2, pp. 47–103). Hillsdale, NJ: Lawrence Erlbaum Associates.

Spearman, C. (1927). *The abilities of man: Their nature and measurement*. London: Macmillan.

Spearman, C. (1923). *The nature of intelligence and the principles of cognition*. London: Macmillan.

Spelke, E. S., & Kinzler, K. D. (2007). Core knowledge. *Developmental Science, 10*(1): 89-96.

Stankiewicz, B. J., & Hummel, J. E. (2002). Automatic priming for translation- and scale-invariant representations of object shape. *Visual Cognition, 9*(6), 719-739.

Stankiewicz, B. J., Hummel, J. E., & Cooper, E. E. (1998). The role of attention in priming for left-right reflections of object images: Evidence for a dual representation of object shape. *Journal of Experimental Psychology: Human Perception and Performance, 24*(3), 732-744.

Sternberg, R. J. (1977). *Intelligence, Information Processing, and Analogical Reasoning*. Hillsdale, NJ: Erlbaum.

Stollinski, R., Schwering, A., Kühnberger, J.-U., & Krumnack, U. (2009). Structural alignment of visual stimuli influences human object categorization. *Proceedings of the 2nd International Analogy Conference*.

Tarr, M. J., Bülthoff, H. H., Zabinski, M., & Blanz, V. (1997). To what extent do unique parts influence recognition across viewpoint? *Psychological Science, 8*(4), 282-289.

Tarr, M. J., Williams, P., Hayward, W. G., & Gauthier, I. (1998). Three-dimensional object recognition is viewpoint dependent. *Nature Neuroscience, 1,* 275-277.

Thoma, V., Hummel, J. E., & Davidoff, J. (2004). Evidence for holistic representations of ignored images and analytic representations of attended images. *Journal of Experimental Psychology: Human Perception and Performance, 30*(2), 257-267.

Thompson, E.P., Chaiken, S., & Hazlewood, J.D. (1993). Need for cognition and desire for control as moderators of extrinsic reward effects: A person X situation approach to the study of intrinsic motivation. *Journal of Personality and Social Psychology*, *64*(6), 987-999.

Treisman, A. M., & Gelade, G. (1980) A feature-integration theory of attention. *Cognitive Psychology, 12*, 97-136.

Trickett, S. B., Trafton, J. G., Saner, L. D., & Schunn, C. D. (2007). "I don't know what is going on there": The use of spatial transformations to deal with and resolve uncertainty in complex visualizations. In M. C. Lovett & P. Shah (Eds.), *Thinking with Data*. Mahwah, NJ: Erlbaum.

Ullman, S. (1984). Visual routines. *Cognition, 18*, 97-159.

van der Ven, A. H. G. S., Ellis, J. L. (2000). A Rasch analysis of Raven's standard progressive matrices. *Peronsality and Individual Differences, 29,* 45-64.

Vodegel-Matzen, L. B. L., van der Molen, M. W., & Dudink, A. C. M. (1994). Error analysis of Raven test performance. *Personality and Individual Differences, 16*(3), 433-445.

Williams, J. E., & McCord, D. M. (2006). Equivalence of standard and computerized versions of the Raven Progressive Matrices Test. *Computers in Human Behavior, 22*, 791-800.

Zagar, R., Arbit, J. & Friedland, J. (1980). Structure of a psychodiagnostic test battery for children. *Journal of Clinical Psychology, 36,* 313 - 318.

Appendix

Appendix A: Modifications to the Structure-Mapping Engine

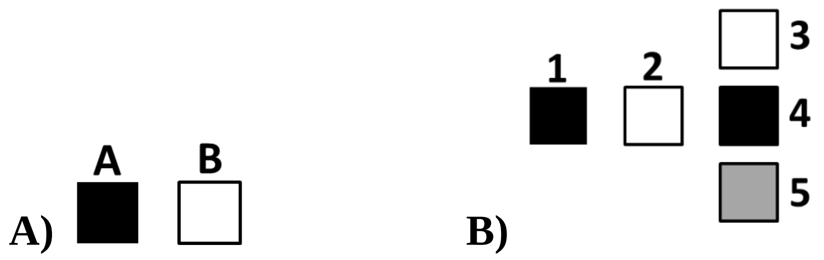


Figure A.1 Two images that might be compared.

Here I describe an SME modification I developed to improve the quality of mappings. I begin with the default behavior (Falkenhainer, Forbus, & Gentner, 1989). Suppose SME is comparing Figure A.1A to 1B. Further suppose that the representations contain only color attributes and two positional relations: **rightOf** and **above**. SME computes mappings in three steps:

1) Form local *match hypotheses* (mh's) between expressions in the base and target. SME identifies every match between expressions with identical predicates. For example, in Figure A.1 (**BlackColored** Object-A) would match to (**BlackColored** Object-1) and (**BlackColored** Object-4). (**rightOf** Object-B object-A) would match to four **rightOf** expressions in the target: (**rightOf** Object-2 Object-1), (**rightOf** Object-3 Object-2), (**rightOf** Object-4 Object-2), and (**rightOf** Object-5 Object-2).

Assign a constant score to each expression mh. Allow these scores to trickle down to mh's between their arguments. For example, consider objects B and 2. There are two expression mh's that have these entities as their arguments: (**WhiteColored** Object-B)⇔(**WhiteColored** Object-2) and (**rightOf** Object-B Object-A)⇔(**rightOf** Object-2 Object-1). Each of these trickles down to the Object-B⇔Object-2 mh.

Thus, for this explanation, we can assign the match a score of 2, indicating the match receives trickle-down from two parents.[10]

Now consider objects A and 2. These objects are different colors. However, they receive trickle down from three expression mh's: (**rightOf** Object-B Object-A) can map to (**rightOf** Object-3 Object-2), (**rightOf** Object-4 Object-2), and (**rightOf** Object-5 Object-2). Therefore, this match has a total score of 3.

This means the highest-scoring entity mh is between Object-A and Object-2, even though these objects are different colors. Essentially, SME likes aligning Object-A with Object-2 because Object-A has an object right of it and Object-2 has several objects right of it. This seems to be in conflict with the intuition that Object-A maps to Object-1 and Object-B maps to Object-2.

2) Group match hypotheses together into *kernels*. Kernels consist of an mh between two expressions, along with mh's between their arguments. For example, one kernel would contain (**rightOf** Object-B Object-A)⇔(**rightOf** Object-3 Object-2), Object-B⇔Object-3, and Object-A⇔Object-2. Each kernel receives a score equal to the sum of its mh's scores.

3) Compute global mappings by finding maximally large sets of consistent kernels. This is implemented via a greedy merge process (Forbus & Oblinger, 1990). It begins by finding the highest-scoring kernel. It incrementally adds in the highest-scoring consistent kernel until no more kernels can be added. It then looks for a secondary mapping.

[10] SME uses small initial activation weights which are multiplied for each level of trickle-down. However, in the present case, where we are considering only first-order relations and attributes, we can simplify this by merely counting the number of parents.

In this case, the highest-score kernel is based on the (**rightOf** Object-B Object-A)⇔(**rightOf** Object-3 Object-2) mh. This kernels receives the highest score because of the strong support for aligning Object-A with Object-2.

I have introduced a modification to SME which prevents it from finding this unintuitive mapping. The key difference is during trickle down. When a match receives trickle down from expression mh's, we prevent it from receiving trickle down from mh's that are inconsistent with each other. This is handled as follows: 1) Sort a match's parents by score. 2) Pick the highest-scoring parent and receive trickle down. 3) Greedily pick additional parents that are consistent with the parents picked thus far and receive trickle down from them.

Consider again Object-A⇔Object-2. This receives trickle down from three **rightOf** mh's, but these mh's are inconsistent. For example, (**rightOf** Object-B Object-A)⇔(**rightOf** Object-3 Object-2) is inconsistent with (**rightOf** Object-B Object-A)⇔(**rightOf** Object-4 Object-2) because one of these aligns Object-B with Object-3 and the other aligns Object-B with Object-4. Because they are inconsistent, Object-A⇔Object-2 only gets credit for one of them—it doesn't matter which one, as they are all worth the same score. It receives trickle down from just one mh, and therefore, it gets a total score of only 1.

Now consider again Object-B⇔Object-2. This receives trickle down from two mh's: (**WhiteColored** Object-B)⇔(**WhiteColored** Object-2) and (**rightOf** Object-B Object-A)⇔(**rightOf** Object-2 Object-1). These two mh's *are* consistent, so the mh receives trickle down from both. It continues to have a total score of 2, meaning it is now a higher-scoring mh than Object-A⇔Object-2.

The above modification is turned on by an optional parameter to SME: **:block-most-out-of-mapping-support?** When this parameter is flipped on, one other change is introduced: after SME finishes computing a global mapping, it recomputes the score for each mh in the mapping, only allowing trickle down from other mh's in the mapping. Thus, it ensures that the final mapping score will be based solely on what aligns in the mapping, rather than receiving trickle down from any mh that doesn't fit the mapping.

Appendix B: The Geometric Analogy Routine

There are two components to the geometric analogy spatial routine: the subroutine for finding differences between two images (Figure 6.3), and the overall routine for solving the problem. They are split up in this way because the overall routine uses the subroutine several times (for A/B differences and for C/answer differences). Thus it would be inefficient to fully script the subroutine each time. Instead, I utilize LISP, the programming language in which the routines are currently implemented. There is a LISP function for producing the subroutine. This function can take different arguments, including the pre-check for the subroutine, its inputs, some of its parameters, and the variable name it will be stored under in the routine working memory. Note, however, that the LISP function is called only to generate the spatial routine. The function is not called during problem-solving.

In the subroutine and full routine, I use the following text style conventions:

Operation type keywords and control structure keywords: ***bold italics***

Global thresholds (e.g., **gt-threshold**): **bold**

LISP function arguments: *italics*

LISP function calls: underlined

Comments (not part of the code): *;;italics preceded by semicolons*

These conventions are made in this appendix for clarity only. They are not inherent to the spatial routine language.

Subroutine for Finding Differences between Images

This subroutine contains several calls to *Find*-Differences. Each *Find-Differences* call corresponds to a node in Figure 6.3. Thus, any time the geometric analogy routine refers to this subroutine, that entire set of strategic choices is performed. Note that the *top-transform* argument specifies which shape transformation should be considered first. Thus, the routine can implement the Rotation shape comparison mode by passing in **:rotation** for this argument, or it can implement the Reflection mode by passing in **:reflection**.

Section 4.4 describes the syntax for operation calls and control structures. Here, the entire subroutine is one large And Statement. The first operation calls *Find-Differences* on the two images. The remaining operation calls update the differences based on various parameters or directives.

```
(defun geo-row-diffs-plan (pre-check inputs name top-transform comment
                           &optional (display? t) (store-var))
 `(,pre-check
   :and :last
   :reset-and-display? ,display? :name ,name
   :store-var ,store-var

   ((t :find-differences
      ,inputs (:pov-type :diffs :encode-added-object-shapes? t
                    :top-transform ,top-transform)
      nil
      :name ,name
      :comment ,comment)

    ((:result :subshape-divided-shapes)
     :find-differences
     :update
     nil (:segment-elements-as-directed (:result :subshape-divided-shapes))
     :comment "Subdivide shapes")

    (t :repeat (:not (:result :partially-matched-shapes))
      :trace? t
      :comment “Break down shapes”
      ((t :find-differences
         :update
         nil (:segment-elements (:result :partially-matched-shapes))
         :comment "Break down shapes")))

    ((:and (:result :elements-broken-down?) (:result :groupable-shapes))
     :find-differences
     :update
     nil (:new-groups (:result :groupable-shapes) :keep-old-constraints? t)
     :comment "Group up shapes")
    )))
```

Main Geometric Analogy Routine

Note that all *Perceive* operations (in every routine) use the parameters described in section 5.1. Here, I refer to these as **perceive-params**.

```
(t :and :last
  ((t :survey
     nil (:sizes? t) nil)

   (t :locate
     (:x 1 :y 1) nil nil
     :store-var :AB-lattice)

   (t :locate
     (:x 2 :y 1) nil nil
     :store-var :C-lattice)

   (t :locate
     (:C-lattice :x 2) nil nil
     :store-var :D-loc
     :comment “Locate the missing image, the second square in the C lattice.”)

   (t :locate
     (:y 2) nil nil
     :store-var :answer-lattice)

    ;;Encode images A, B, and C, and the answer images
    (t :perceive
     (:AB-lattice :x 1)
     ,perceive-params nil
     :name "Image A"
     :store-var :A)

   (t :perceive
     (:AB-lattice :x 2)
     ,perceive-params nil
     :name "Image B"
     :store-var :B)

   (t :perceive
     (:C-lattice :x 1)
     ,perceive-params nil
     :name "Image C"
     :store-var :C)

   (t :perceive
     (:ans-lattice)
     ,perceive-params nil
     :name "Image "
     :store-var :answers)

   ;;Iterate over the two overall strategies.  Take the first one that gives a unique, sufficient answer.
   (t
    (:iterate-over ("Visual Inference" "Second-Order Comparison") :strategy)
    (:first (:and :single-value?
                 (:or (:not (:result :differences-detected?))
                      (:min-score ,min-mapping-score)))
           :best-score)
    :store-var :answer-gen
    :store-input :best-strategy
    :reset-and-display? t
    :trace? t
    :comment "Try different strategies"
    :iteration-comment :strategy

    ;;Do the following if we're trying the visual inference strategy.
    (((:eq :strategy "Visual Inference")
     :and :contingent
     :post-check (:and :single-value?
                       (:or (:not (:result :differences-detected?))
                            (:min-score ,min-mapping-score)))
     ;;Find the differences between A and B
     (,(geo-row-diffs-plan t `(:A :B) "Diffs(A,B)" :identity
                    "Find differences between Images A and B"
                    t :AB-diffs)

       ;;Apply the differences to C to infer D'
       (t :infer-image (:AB-diffs :C :D-loc)
         nil nil
         :store-var :D
         :name "Image D"
         :comment (:string "Solve for D: "
                      (:or (:and (:result :apply-problems) "Failed")
                           (:and (:not (:result :apply-problems)) "Succeeded"))))

       ;;Compare the inferred D' to each possible answer
       ((:not (:result :apply-problems :D))
        (:iterate-over :answers :answer)
        (:first (:not (:result :differences-detected?))
              :best-score :forced-choice)
        :reset-and-display? t
        :store-input :best-answer
        :trace? t
        :comment "Try each answer"
        :iteration-comment (:string "Try answer: " :answer)

        ((t :find-differences
         (:D :answer)
         (:pov-type :diffs :ignore-for-attributes? t))
         nil
         :store-var :D-answer-diffs
         :comment "Compare to D")

       (t :repeat (:not (:result :subshape-divided-shapes))
         :trace? t
         :comment "Subdivide shapes"
         (((:result :subshape-divided-shapes)
           :find-differences :update
           nil
           (:segment-elements-as-directed (:result :subshape-divided-shapes))
           :store-var :D-answer-diffs
           :comment "Subdivide shapes")))
       ))))


   ((:eq :strategy "Second-Order Comparison")
    :abandon (:D) nil nil)

   ;;Do the following if we're trying second-order comparison.
   ;;Iterate over the four mapping modes for the A/B image comparison
   ((:eq :strategy "Second-Order-Comparison")
    (:iterate-over (:identity :reflection :rotation "Try secondary mapping") :AB-comp)
    (:first (:and :single-value?
                      (:or (:not (:result :differences-detected?))
                           (:min-score ,min-mapping-score)))
         :best-score)
    :store-input :best-AB-comp
    :reset-and-display? t
    :trace? t
    :comment "Try different A->B mappings"
    :iteration-comment :AB-comp

    ((t :and :contingent
       ((t :or :last
         :reset-and-display? t
         :store-var :AB-diffs
         ;;The following implements the Normal, Rotation, or Reflection modes
         ;;(depending on the value of :AB-comp).
         (,(geo-row-diffs-plan
           `(:and (:not (:eq :AB-comp "Try secondary mapping"))
                  (:or (:eq :AB-comp :identity)
                       (:and (:eq (:result :num-elements :A) 1)
                             (:eq (:result :num-elements :B) 1))))
           `(:A :B)
           "Diffs(A,B)"
           :AB-comp
           "Find differences between Images A and B")
```

```
   ;;The following implements the Alt-Mapping or secondary-mapping mode.
   ((:eq :AB-comp "Try secondary mapping")
    :find-differences
    (:update :AB-diffs)
    (:pov-type :diffs :shape-feature-constrained? nil
            :encode-added-object-shapes? t)
    (:try-different-mappings? t)
    :name "Diffs(A,B)"
    :comment "Find differences between Images A and B")))

;;Now iterate over the same mapping modes for the C/answer comparison
(t
 (:iterate-over (:identity :reflection :rotation "Try secondary mapping") :C-ans-comp)
 (:first (:and :single-value?
              (:or (:not (:result :differences-detected?))
                   (:min-score ,min-mapping-score)))
     :best-score)
 :store-input :best-C-ans-comp
 :trace? t
 :comment "Try different C->answer mappings"
 :iteration-comment :C-ans-comp

 ((t
   (:iterate-over :answers :answer)
   (:best-score :forced-choice)
   :reset-and-display? t
   :store-input :best-answer
   :trace? t
   :comment "Try each answer"
   :iteration-comment (:string "Try answer: " :answer)

   ((t :and :contingent
      ((t :or :last
         :reset-and-display? t
         :store-var :C-ans-diffs
         :store-under :answer

         (,(geo-row-diffs-plan
           `(:and (:not (:eq :C-ans-comp "Try secondary mapping"))
                  (:or (:eq :C-ans-comp :identity)
                       (:and (:eq (:result :num-elements :C) 1)
                             (:eq (:result :num-elements :answer) 1))))
           `(:C :answer)
           "Diffs(C,answer)"
           :C-ans-comp
           "Find diffs between C and the possible answer")

           ((:eq :C-ans-comp "Try secondary mapping")
            :find-differences
```

```
                    (:update (:stored-under :answer :C-ans-diffs))
                    (:pov-type :diffs :shape-feature-constrained? nil
                            :encode-added-object-shapes? t)
                    (:try-different-mappings? t)
                    :name "Diffs(C,answer)"
                    :comment "Find diffs between Image C and the possible answer")))

                (t :find-differences
                  (:AB-diffs (:stored-under :answer :C-ans-diffs)) nil nil
                  :name "Answer Comparison"
                  :comment "Compare the two sets of differences"))))))))))))

   ;;Find the answer image.  It will be one of the arguments of the generalization
   ;:answer-gen.  It must be retrieved slightly differently depending on which strategy was used.
   ((:and (:stored? :answer-gen) (:eq :best-strategy "Apply differences"))
    :access (:input 2 :answer-gen)
    nil nil
    :store-var :answer)
   ((:and (:stored? :answer-gen) (:eq :best-strategy "Compare differences"))
    :access (:input 2 (:input 2 :answer-gen))
    nil nil
    :store-var :answer)

   (t :select (:answer))))
```

Appendix C: The Raven's Progressive Matrices Routine

As with geometric analogy, this routine is divided into a subroutine for finding differences and the overall routine. The same style conventions as above are used. See section 4.4 for details on the spatial routine syntax.

Subroutine for Finding Differences across a Row of Images

The subroutine is given below. As in Appendix B, it is written as a LISP function. The inputs to the function include *init-store-var*. The initial pattern of variance will be stored under this name. This is important because when the routine implements alternate strategies (i.e., literal patterns of variance), it bases them off the initial pattern of variance.

Each call to *Find-Differences* corresponds to a node in Figures 7.7-7.8.

```
(defun raven-row-diffs-plan (pre-check inputs store-var init-store-var name &key (display? t)
                             (break-down-check t))
  `(,pre-check
   :and :last
   :store-var ,store-var
   :reset-and-display? ,display? :name ,name

   ((t :find-differences
      ,inputs (:pov-type :diffs)
      nil
      :name ,name
      :store-var ,init-store-var
      :comment "Initial differences")

    ((:result :subshape-divided-shapes)
     :find-differences :update
     nil (:segment-elements-as-directed (:result :subshape-divided-shapes))
     :comment "Subdivide shapes")

    (,break-down-check
     :repeat (:not (:result :partially-matched-shapes))
     :trace? t
     :comment "Break down shapes"
     ((t :find-differences :update
        nil (:segment-elements (:result :partially-matched-shapes))
        :comment "Break down shapes")))

    ((:result :first-to-last-match-shapes)
     :find-differences :update
     (:first-to-last? t)
     (:constraining-strict-shape-types (:result :all-perfect-match-strict-shapes))
     :comment "Compare the first and last images")

    ((:and (:result :elements-broken-down?) (:result :groupable-shapes))
     :find-differences :update
     nil
     (:new-groups (:result :groupable-shapes) :keep-old-constraints? t)
     :comment "Group up shapes")

    ((:result :better-mappings-for-mismatched-objects)
     :find-differences :update
     nil
     (:preferred-mappings (:result :better-mappings-for-mismatched-objects))
     :comment "Use a secondary mapping")

    ((:and (:parameter :first-to-last?) (:result :shape-differences?))
 :find-differences :update
 (:strict-shape-constrained? t)
 nil
 :comment "Apply strict shape matches across the board")
)))
```

Main Raven's Matrices Routine

As above, all *Perceive* operations use the parameters described in section 5.1. Here, I refer to these as **perceive-params**.

```
(t :and :last
  ((t :survey
     nil (:sizes? t :line-widths? t) nil)

   (t :locate
     (:y 1) nil nil
     :store-var :prob-lattice)

   (t :locate
     (:y 2) nil nil
     :store-var :ans-lattice)

   ;;Let's find out how many rows and columns the matrix has…
   (t :inspect
     (:prob-lattice) nil nil
     :store-var :prob-lattice-info)

   ;;Find the location of the missing image, i.e., the right/bottom-most one
   (t :locate
     (:prob-lattice :x (:result :num-cols :prob-lattice-info)
                    :y (:result :num-rows :prob-lattice-info))
     nil nil
     :store-var :answer-loc)

   ;;First, see if we can just solve this via Texture Completion.  Only if it's a 1x1 matrix…
   ((:lt (:result :num-rows :prob-lattice-info) 2)
    :and :last

    ((t :locate
       (:prob-lattice :x 1 :y 1 "Problem" :y 1 :referent "Answer" :x 2 :y 2)
       nil nil
       :store-var :upper-corridor)

     (t :detect-texture
    (:upper-corridor) nil nil
    :store-var :upper-corridor-texture)

  ;;Did we detect a texture along the upper corridor?  If so, see if we can use an answer
  ;;along the lower corridor.
  ((:stored? :upper-corridor-texture)
   :and :last
   ((t :locate
      (:prob-lattice :x 1 :y 1 "Problem" :y 2 :referent "Answer" :x 2 :y 2)
      nil nil
      :store-var :lower-corridor)

    (t :locate
      (:prob-lattice :x 1 :y 1 "Answer")
      nil nil
      :store-var :reference-loc)

    (t :locate
      (:ans-lattice)
      (:fully-specified-only? t) nil
      :store-var :answer-locs)

    ;;Insert each possible answer
    (t (:iterate-over :answer-locs :answer) (:best-score :forced-choice)
      :store-input :answer
      :store-var :answer-window
      ((t :detect-texture
          (:bottom-row)
          (:min-auto-corr .2)
          (:target-size (:result :window-size :top-row-texture)
                        :insertion-referent (:reference-loc)
                        :insertion-target (:answer)))))))))

;;If Texture Completion worked, an answer will stored as :answer.  If not, we'll have to try
;;something else.  That means visual inference or second-order comparison.

;;Locate & encode the top row of a 1x1 matrix….
((:and (:not (:and (:stored? :answer) (:single-value? :answer)))
          (:lt (:result :num-rows :prob-lattice-info) 2))
 :and :last
 :store-var :top-row-reps
 ((t :perceive
    (:prob-lattice :x 1 :y 1 "Problem" :x 1 :y 1 :referent "Answer" :x 2 :y 2)
    ,perceive-params nil
    :store-var :top-left-rep :name "Image 1,1")

  (t :perceive
    (:prob-lattice :x 1 :y 1 "Problem" :x 2 :y 1 :referent "Answer" :x 2 :y 2)
    ,*first-raven-vrep-params* nil
    :store-var :top-right-rep :name "Image 1,2")
  (t :access
     (:top-left-rep :top-right-rep) nil nil)))


;;Locate & encode the bottom row of a 1x1 matrix…
((:and (:not (:and (:stored? :answer) (:single-value? :answer)))
           (:lt (:result :num-rows :prob-lattice-info) 2))
 :perceive
 (:prob-lattice :x 1 :y 1 "Problem" :x 1 :y 2 :referent "Answer" :x 2 :y 2 :rows 2 :cols 2)
 ,perceive-params nil
 :store-var :bottom-row-reps :name "Image 2,1")

;;Locate & encode the top row of a larger matrix
((:gt (:result :num-rows :prob-lattice-info) 1)
 :perceive
 (:prob-lattice :row 1)
 ,perceive-params nil
 :store-var :top-row-reps :name "Image 1,"
 :comment "Initial encoding")

;;Locate & encode the middle row, if there are more than two rows
((:gt (:result :num-rows :prob-lattice-info) 2)
 :perceive
 (:prob-lattice :row 2)
 ,perceive-params nil
 :store-var :mid-row-reps :name "Image 2,"
 :comment "Initial encoding")

;;Locate & encode the bottom row of a larger matrix
((:gt (:result :num-rows :prob-lattice-info) 1)
 :perceive
 (:prob-lattice :row (:result :num-rows :prob-lattice-info))
 ,perceive-params nil
 :store-var :bottom-row-reps :name "Image 3,"
 :comment "Initial encoding")

;;Finished encoding…


;;Encode the pattern of variance for the completed rows of the matrix.  This means:
;;1) Encode the top row
;;2) Encode the middle row (if there are three rows)
;;3) Compare the top and middle rows.  If they don't align, select a different strategy for
;;    representing patterns of variance (e.g., literal or simple)
((:not (:and (:stored? :answer) (:single-value? :answer)))
 :and (:first-score ,min-mapping-score :best-score)
 :store-var :row-gen
 :reset-and-display? t :name "Row-Gen"

 ;;First try a normal differences pattern of variance for the two rows…
```

```
((t :and :last
    :store-var :row-gen
    :reset-and-display? t :name "Row-Gen"

    (,(raven-row-diffs-plan t :top-row-reps :top-row-diffs :top-row-init-diffs "Top Row")

     ,(raven-row-diffs-plan
        `(:gt (:result :num-rows :prob-lattice-info) 2)
        :mid-row-reps :mid-row-diffs :mid-row-init-diffs "Mid Row")

      ((:gt (:result :num-rows :prob-lattice-info) 2)
       :generalize
       (:top-row-diffs :mid-row-diffs)
       nil nil
       :name "Top Rows"
       :comment "Evaluate the top two rows")))

 ;;Now try a normal, literal pattern of variance (only if there is a middle row)
 ((:gt (:result :num-rows :prob-lattice-info) 2)
  :and :contingent
  :post-check (:not (:result :differences-detected?))
  :store-var :row-gen
  :reset-and-display? t :name "Row-Gen"
  :trace? t
  :comment "Try a literal rep on the top rows"

  ((t :find-differences (:update :top-row-init-diffs)
     (:pov-type :literal) (:keep-old-constraints? t)
     :store-var :literal-top-row-diffs
     :reset-and-display? t :comment "Literal rep")

   (t :find-differences (:update :mid-row-init-diffs)
     (:pov-type :literal) (:keep-old-constraints? t)
     :store-var :literal-mid-row-diffs
     :reset-and-display? t :comment "Literal rep")

   (t :generalize
     (:literal-top-row-diffs :literal-mid-row-diffs)
     nil nil
     :name "Top Rows")))

 ;;Now try a simple literal pattern of variance
 ((:gt (:result :num-rows :prob-lattice-info) 2)
  :and :contingent
  :post-check (:and
                  (:result :destructive-pov-mapping?)
                  (:not (:result :differences-detected?)))
  :store-var :row-gen
  :reset-and-display? t :name "Row-Gen"
  :trace? t
```

```
:comment "Try a literal, simple rep on the top rows"

(((:stored? :literal-top-row-diffs)
  :find-differences (:update :literal-top-row-diffs)
  (:pov-type :literal :simple? t) (:keep-old-constraints? t)
  :store-var :simple-top-row-diffs
  :reset-and-display? t :comment "Literal, simple rep")

 ((:stored? :literal-mid-row-diffs)
  :find-differences (:update :literal-mid-row-diffs)
  (:pov-type :literal :simple? t) (:keep-old-constraints? t)
  :store-var :simple-mid-row-diffs
  :reset-and-display? t :comment "Literal, simple rep")

 (t :generalize
    (:simple-top-row-diffs :simple-mid-row-diffs)
    nil nil
    :name "Top Rows")))

;;Now try a simple, differences pattern of variance
((:gt (:result :num-rows :prob-lattice-info) 2)
 :and :contingent
 :post-check (:result :destructive-pov-mapping?)
 :store-var :row-gen
 :reset-and-display? t :name "Row-Gen"
 :trace? t
 :comment "Try a simple rep on the top rows"

 ((t :find-differences (:update :top-row-diffs)
    (:pov-type :diffs :simple? t) (:keep-old-constraints? t)
    :store-var :top-row-diffs
    :reset-and-display? t :comment "Simple rep")

  (t :find-differences (:update :mid-row-diffs)
    (:pov-type :diffs :simple? t) (:keep-old-constraints? t)
    :store-var :mid-row-diffs
    :reset-and-display? t :comment "Simple rep")

  ;;When comparing patterns of variance, first force a non-destructive mapping, in which
  ;;objects in the same image map to objects in the same image.  If the destructive mapping
  ;;can't score higher than this non-destructive one, then we didn't need simple patterns in the
  ;;first place.
  (t
   :or (:best-score :forced-choice)
   :post-check (:and (:single-value?) (:not (:parameter :strict-ordering?)))

   ((t :generalize
      (:top-row-diffs :mid-row-diffs)
      ,(add-param :strict-ordering? t *raven-gen-params*) nil
      :name "Top Rows"
```

```
        :comment "Strict ordering enforced")

      (t :generalize
        (:top-row-diffs :mid-row-diffs) ,*raven-gen-params* nil
        :name "Top Rows"
        :comment "Do we get a higher score without strict ordering?")))))))

;;Finished finding a pattern of variance for the rows

 ;;Encode the answer images.  This uses an Or Statement because there are slightly different
 ;;different encoding rules for 1x1 matrices.
((:not (:and (:stored? :answer) (:single-value? :answer)))
 :or :last
 (((:gt (:result :num-rows :prob-lattice-info) 1)
   :perceive
   (:ans-lattice)
   ,perceive-params nil
   :name "Image "
   :store-var :answer-reps
   :comment "Initial encoding")

  ;;This uses a directive indicating we want to pick out the biggest object and then find everything
  ;;inside it.
  ((:lt (:result :num-rows :prob-lattice-info) 2)
   :perceive
   (:ans-lattice)
   ,perceive-params (:location-details (:biggest))
   :name "Image "
   :store-var :answer-reps
   :comment "Initial encoding")))

;;Solve for the answer via Visual Inference
((:and (:not (:and (:stored? :answer) (:single-value? :answer)))
          (:stored? :row-gen))
 :and :last

 ;;Compute a pattern of inference for the known images in the bottom row (assuming this is 3x3).
 ;;We'll then update this so that it's encoded using the same strategy as the above rows.
 (,(raven-row-diffs-plan
    `(:gt (:result :num-cols :prob-lattice-info) 2)
    :bottom-row-reps
    :bottom-row :bottom-row-init "Bottom Row"
    :break-down-check nil
    )

    ;;Update to first-to-last
    ((:parameter :first-to-last? :row-gen)
     :find-differences
     (:update :bottom-row)
     (:strict-shape-constrained? t
                            :always-encode-shape-constant? t)
     nil
     :store-var :bottom-row
     :comment "Require strict shape matches")

    ;;Update to literal (possibly simple literal)
    ((:eq (:parameter :pov-type :row-gen) :literal)
     :find-differences
     (:update :bottom-row-init)
     (:pov-type (:parameter :pov-type :row-gen) :simple? (:parameter :simple? :row-gen))
     (:keep-old-constraints? t)
     :store-var :bottom-row
     :comment "Encode literal differences"
     :name "Bottom")

     ;;Update to simple (simple differences)
    ((:and (:eq (:parameter :pov-type :row-gen) :diffs) (:parameter :simple? :row-gen))
     :find-differences
     (:update :bottom-row)
     (:pov-type (:parameter :pov-type :row-gen) :simple? t
            :always-encode-shape-constant? t)
     (:keep-old-constraints? t)
     :store-var :bottom-row
     :comment "Use simple differences"
     :name "Bottom")

    ;;If this isn't a 3x3, just store the bottom-row images, rather than a pattern of variance.
    ((:lt (:result :num-cols :prob-lattice-info) 3)
     :access :bottom-row-reps nil nil
     :store-var :bottom-row)

    (t :infer-image (:row-gen :bottom-row :answer-loc)
       nil nil
       :store-var :D
       :name "Answer Image"
       :post-check (:not (:result :apply-problems))
       :comment (:string "Solve for answer: "
                    (:or (:and (:result :apply-problems) "Failed")
                        (:and (:not (:result :apply-problems)) "Succeeded"))))))

;;Finished inferring the answer image

;;If we inferred the answer, compare it to each possible answer
((:and (:not (:and (:stored? :answer) (:single-value? :answer)))
         (:stored? :row-gen) (:stored? :D))
 (:iterate-over :answer-reps :answer)
 (:first (:not (:result :differences-detected?))
       :best-score :forced-choice)
 :reset-and-display? t
 :store-var :answer-gen
```

```
:store-input :answer
:store-second-var :second-answer-gen
:store-second-input :second-answer
:trace? t
:comment "Try each answer"
:iteration-comment (:string "Try answer: " :answer)

;;Make sure to ignore any features that weren't inferred
((t :find-differences
    (:D :answer)
    (:pov-type diffs :ignore-for-attributes? t
                     :ignore-spatial-relations? (:parameter :simple? :bottom-row)
                     :strict-shape-biased? (:result :attributes-only? :bottom-row))
    nil
    :store-var :D-answer-diffs
    :comment "Compare to D")

 ;;Do simple perceptual reorganization if necessary
 (t :repeat (:not (:result :subshape-divided-shapes))
    :trace? t
    :comment "Subdivide shapes"
    (((:result :subshape-divided-shapes)
      :find-differences :update
      nil
      (:segment-elements-as-directed (:result :subshape-divided-shapes))
      :store-var :D-answer-diffs
      :comment "Subdivide shapes")))))

;;Finished with visual inference strategy

;;If Visual Inference failed, try Second-Order Comparison: Insert each possible answer
;;into the bottom row.
((:and (:stored? :row-gen)
       (:not (:and (:stored? :answer) (:single-value? :answer)))
       (:not (:stored? :D)))
 (:iterate-over :answer-reps :answer) (:best-score :forced-choice)
 :reset-and-display? t
 :store-var :answer-gen
 :store-input :answer
 :store-second-var :second-answer-gen
 :store-second-input :second-answer
 :trace? t
 :comment "Try each answer"
 :iteration-comment (:string "Try answer: " :answer)

 ;;Insert the answer into the bottom row, compute a pattern of variance
 (,(raven-row-diffs-plan t `(:bottom-row-reps :answer)
                    :bottom-row-diffs :bottom-row-init-diffs "Bottom" :display? nil)

   ;;Now update the pattern so it uses the same strategy as the above rows.
```

```
        ((:eq (:parameter :pov-type :row-gen) :literal)
         :find-differences (:update :bottom-row-init-diffs)
         (:pov-type (:parameter :pov-type :row-gen) :simple? (:parameter :simple? :row-gen))
         (:keep-old-constraints? t)
         :store-var :bottom-row-diffs
         :name "Bottom")

        ((:and (:eq (:parameter :pov-type :row-gen) :diffs) (:parameter :simple? :row-gen))
         :find-differences (:update :bottom-row-diffs)
         (:pov-type (:parameter :pov-type :row-gen) :simple? (:parameter :simple? :row-gen))
         (:keep-old-constraints? t)
         :store-var :bottom-row-diffs
         :name "Bottom")

        ;;Now compare this bottom row pattern to the above rows to evaluate the answer.
        (t :generalize
           (:row-gen :bottom-row-diffs)
           nil nil
           :store-var :answer-gen
           :name "Answer-Gen"
           :comment "Evaluate the answer")))

    ;;Finished trying out each answer

    (t :select (:answer))))
```

Appendix D: The Oddity Task Routine

This routine uses the same style conventions as above (see Appendix B). See section 4.4 for details on the spatial routine syntax.

As above, all *Perceive* operations use the parameters described in section 5.1. Here, I refer to these as **perceive-params**.

```
(t :and :last
  ((t :survey
      nil (:sizes? t :line-widths? t) nil)

   (t :locate
      (:y 1) nil nil
      :store-var :top-row)

   (t :locate
   (:y 2) nil nil
   :store-var :bottom-row)

;;Top-level strategic choice: How do we compute similarity?  :base coverage identifies odd images
;;out that lack a feature, while :symmetric coverage can identify an image with an extra feature.
(t
 (:iterate-over (:base :symmetric) :coverage-type) (:first :single-value?)
 :reset-and-display? t
 :store-var :answer-gen
 :trace? t
 :comment "Try different similarity measures"
 :iteration-comment :coverage-type

 ;;Next strategic choice: do we encode at the group or edges level?
 ((t
   (:iterate-over (:groups :edges) :encoding-focus) (:first :single-value?)
   :reset-and-display? t
   :store-var :answer-gen
   :trace? t
   :comment "Try different levels of representation"
   :iteration-comment :encoding-focus


   ;;Encode top row
   (((:not (:eq :encoding-focus :edges))
     :perceive
     (:top-row)
     ,perceive-params
     nil
     :name "Image 1,"
     :store-var :top-row-reps)

    ((:eq :encoding-focus :edges)
     :perceive
     (:update :top-row-reps)
     (:focus :encoding-focus)
     nil
     :store-var :top-row-reps)


    ;;Encode bottom row
    ((:not (:eq :encoding-focus :edges))
     :perceive
     (:bottom-row)
     ,perceive-params
     nil
     :name "Image 2,"
     :store-var :bottom-row-reps)

    ((:eq :encoding-focus :edges)
:perceive
(:update :bottom-row-reps)
(:focus :encoding-focus)
nil
:store-var :bottom-row-reps)

;;Finished encoding

;;Before we continue, make sure that every image was successfully encoded at the right level.
;;For example, if we're doing the edge level and some images have multiple objects, perception
;;will have failed.
((:and
  (:every (:eq (:parameter :focus) :encoding-focus) :top-row-reps)
  (:every (:eq (:parameter :focus) :encoding-focus) :bottom-row-reps)
  (:every (:gt (:result :num-elements) 0) :top-row-reps)
  (:every (:gt (:result :num-elements) 0) :bottom-row-reps))
 :and :contingent

 ;;See if we need to filter out orientation-specific or size-specific features
 ((t :and :last
    :trace? t
    :comment "Filter out orientations and sizes?"

    ;;Initialize these variables to true, so initially we assume orientations and sizes are fine
    ((t :access t nil nil
       :store-var :good-orientations?)
     (t :access t nil nil
       :store-var :good-sizes?)

     ;;Compute a pattern of variance across the top row and bottom row together.  We'll then
     ;;check for non-adjacent differences of the various types.
     (t
      :find-differences (:top-row-reps :bottom-row-reps)
      (:shape-types? nil :first-to-last? t)
      nil
      :store-var :image-diffs
      :universal-trace? t
      :comment (:string "Orientation diffs: " (:result :nonadjacent-orientation-differences?)
                  " | Size/Shape diffs: "
                  (:result (:nonadjacent-differences-of-types :size :shape))))

     ;;Were there orientation differences? If so, set :good-orientations? to nil.
     ((:result :nonadjacent-orientation-differences? :image-diffs)
      :access nil nil nil
      :store-var :good-orientations?)

     ;;Were there size/shape differences?  If so, set :good-sizes? to nil.
     ((:result (:nonadjacent-differences-of-types :size :shape) :image-diffs)
      :access nil nil nil
      :store-var :good-sizes?)
     ;;Now update the image representations, filtering out orientations or sizes if necessary.
     ((:or (:not :good-orientations?)
            (:not :good-sizes?))
      :and :last
      :trace? t
      :comment "Update to orientation-invariant or size-invariant reps"

      ((t :perceive (:update :top-row-reps)
         (:orientation-specific? :good-orientations? :size-specific? :good-sizes?) nil
         :store-var :top-row-reps)
       (t :perceive (:update :bottom-row-reps)
         (:orientation-specific? :good-orientations? :size-specific? :good-sizes?) nil
         :store-var :bottom-row-reps)))))

  ;;Finished filtering out information

  ;;Iterate over the top and bottom rows.  For each row, build a generalization for that row and
  ;;compare the images in the other row to it.
  (t
   (:iterate-over (1 2) :row-num) (:worst-score :forced-choice :!prefer-single)
   :reset-and-display? t
   :store-var :answer-gen
   :trace? t
   :comment "Try each row"
   :iteration-comment (:string "Try row: " :row-num)

   ((t :and :contingent
      ;;Identify the row for generalizing
      ((t :or :last
         (((:eq :row-num 1)
           :access
           :top-row-reps nil nil
           :store-var :row)
          ((:eq :row-num 2)
          :access
           :bottom-row-reps nil nil
           :store-var :row)))

      ;;Identify the row of answers to compare to the generalization
      (t :or :last
         (((:eq :row-num 1)
           :access
           :bottom-row-reps nil nil
           :store-var :answers)
          ((:eq :row-num 2)
           :access
           :top-row-reps nil nil
           :store-var :answers)))
```

```
            (t :generalize
               :row (:coverage :coverage-type)
               nil
               :reset-and-display? t
               :store-var :row-gen
               :name "Row Gen"
               :comment "Generalize over row")

            ;;Now compare each possible answer to the generalization
            (t
             (:iterate-over :answers :answer) (:worst-score :forced-choice)
             :reset-and-display? t
             :store-var :answer-gen
             :trace? t
             :comment "Try each answer"
             :iteration-comment (:string "Try answer: " :answer)
             ((t :generalize
                 (:row-gen :answer)
                 (:coverage :coverage-type) nil
                 :store-var :answer-gen
                 :name "Answer Gen"
                 :comment (:string "Compare to: " :answer)))))))))))))

;;Finished solving

;;:answer-gen should hold the comparison between the row and best answer.
;;Thus, the best answer will be its second input.
((:stored? :answer-gen)
 :access
 (:input 2 :answer-gen) nil nil
 :store-var :answer)


((:stored? :answer)
  :select (:answer))))
```