NORTHWESTERN UNIVERSITY

Toward Automated Sketching Collaborators: An Analogical Route

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Matthew David McLure

EVANSTON, ILLINOIS

March 2019

ABSTRACT

Toward Automated Sketching Collaborators: An Analogical Route

Matthew McLure

Automated sketch collaborators might help us create more dynamic intelligent tutoring systems, work out designs, reduce bias in solving spatial social problems, and organize our ideas. Here, we examine some properties of sketch recognition methods designed to help serve that goal. Structure Mapping techniques are applied to symbolic structural descriptions of sketched objects in order to classify them. First, we evaluate processes for *encoding* ink into a structural description in ways that are controllably tractable. We contribute a perceptual organization scheme involving *edge-cycles*, which is more abstract, concise, and effective than using edges. We also provide filtering strategies for keeping input tractable for analogical matching, which can be applied to edge-cycles, edges or both, and we demonstrate the value of a hybrid encoding with a filter based on ink-coverage. Second, we evaluate two variants of an approach for extending similarity-based classifiers with classification criteria gleaned from *near-miss* comparisons – highly similar positive-negative pairs – and we look at its performance over a range training data compression rates. We find that in moderation, criteria from near-misses can improve similarity-based classification. Third, we demonstrate that an alternative approach to learning concept boundaries, which uses linear support vector machines built on analogical generalization, achieves higher, more stable accuracy than near-miss and similarity-based classifiers with compression of training data that is equal or better. Finally, we discuss how the analogical approach compares to related work and close by suggesting future work that that could build on these contributions toward the goal of sketching collaborators.

Acknowledgements

Some people make hard things look easy. This body of work was a hard thing that I made look hard. I made it through because I was fortunate enough to be surrounded by supportive and brilliant colleagues, family and friends. If these acknowledgements seem long, know that this was the short version. One can build up a lot of gratitude in a decade.

First, I want to thank the Air Force Office of Scientific Research, which funded this work and brought my ideas to fruition.

I want to especially thank my advisor and mentor Ken Forbus. From my beginning in the Qualitative Reasoning Group (QRG), when science felt most like an uphill battle, Ken felt like a general that would join me in a foxhole. Ken is enthusiastic down to the parentheses, which made for conversations and hacking sessions that were reliably orienting and reinvigorating. Even after I had wandered into the wilderness, Ken never gave up on helping me find my way out. I am grateful for this, most of all.

I thank the rest of my thesis committee, Bryan Pardo, Chris Riesbeck and Doug Downey, for the questions that made me think and the rigor that challenged me. Before serving on my committee, these three educators shaped an enormous swath of my AI education.

Tom Hinrichs and Madeline Usher were two generous resources during my time at QRG. I knocked on their doors enough times to test the hinges. There is a cleverness to both of their work that I hoped would rub off on me if I rolled around in it. This inevitably left messes behind, and I am grateful for their patience in helping clean it up.

From beginning to end, my fellow QRG students were cherished colleagues and friends. I am grateful to Scott Friedman for putting up with me chasing him onto the intercampus shuttle

so many nights in those first couple years, firing off endless questions to replace his podcasts. He planted seeds of advice that took root and helped me get by years later. Scott also helped spark and foster many of the ideas in Chapters 4 and 5 of this thesis. Maria Chang was my unofficial QRG buddy from the day we both visited and agreed on the way back from dinner that QRG seemed special. She made every leap before I did over the years, demonstrating that it was survivable. A fearless leader, I'm not sure if she realizes how precious that was to me. Jason Taylor was a blast of a friend to have in those early years in Chicago (and still is). As Graphs, he also played a significant role in the final stages of my journey, as part of the soundtrack to my writing. My memories of early QRG life with these three are pure treasure.

Subu Kandaswamy was my longest-running office mate, my daily comic relief, and my go-to partner for hunkering down with a six pack and a whiteboard. Subu's impact is felt throughout this body of work, from his contributions to the segmentation algorithm and qualitative descriptors in Section 2.2 to his intuitions that took long enough to sink in that they ended up shaping elements of the future work in Section 7. Andrew Lovett is one of the giants whose shoulders this dissertation was built on, as revealed by a word search for "Lovett". Andrew is a scientist's scientist. Losing debates to him was one of my most valued grad school pastimes. Speaking of giants, Chen Liang is a privilege to have had so much exposure to. He came into QRG sharing my interest in analogical learning and rapidly flourished, opening my mind to what the term could mean. I left every conversation feeling enlightened.

Jon Wetzel was my official QRG buddy. He got me off the ground with generosity and hyper-competence, and he brightened the halls with positivity. Dave Barbella was the reason I had any clue how to assemble a dissertation, and his hilarious perspectives made my days lighter.

I am also grateful to Irina Rabkina, a generous colleague and a quick study. Kezhen Chen arrived when I departed to work remotely, but even from afar, he has been helpful and acutely inquisitive. Irina and Kezhen contributed to this dissertation in a very concrete way, importing the MNIST dataset into CogSketch's polyline format through a computer vision pipeline. Jason Wilson, Joe Blass, CJ McFate and Max Crouse are other gifted QRG researchers whose knowledge shed much-needed light on important topics in AI and cognitive science.

Michael Lucas was my semi-daily source for enlightening conversations about machine learning and computer science theory topics outside of the QRG universe. He was also my partner in crime. Years of long Wednesday nights playing arcade games and pool are some of the fondest Chicago memories I have. My PhD student counterparts at the University of Chicago were a reliable source of good times on the weekends. I am especially grateful to Dan Zou, my roommate and close friend whose statistical expertise my work directly benefited from, and Soraya Lambotte, a pal that was always there to celebrate and commiserate when I needed it. Her wisdom beyond her years and her laugh were comforting. I thank Eliot Baker and Wei Shi, my fellow adventurers, with whom vacations were the ultimate reboot. I thank Vinh Tran and Felix Malinkevich, my old friends, who got me through the final years when I was off the grid.

I want to thank my mom Karin McLure, whose indirect influence on this work spanned 31 years of being there for me every damn time (including at 3 AM). The quintessential super-mom, she has never left my corner. Without her, the opportunities in my life would not have been what they were, my passion for engineering would not be what it is, and my final dissertation defense practice talk wouldn't have had an audience. I want to thank my dad David McLure, whose expertise and infectious interest in computer science inspired my own, and whose creativity and independent mindedness I try to emulate. I thank my sister Amanda, who

got the super-hero gene and has shown me a lot about being an adult over the past decade despite being five years younger. I thank my grandparents. Gail McLure has been a treasured source of wisdom about writing, academia and serenity. John McLure actively ignited my curiosity about science from a young age. Shirley Knope is a relentless depression-era fighter that showed me the value of a struggle, and Robert Knope showed me the intrinsic reward in building something well. I am grateful to my second mother Martha Hosey, the most generous person I know, and to John and Irene Hosey, who were pulling for me in fewer words (always appreciated by PhD students). I am also grateful to Hazel Hosey, our self-trained emotional support dog, who knows when to beckon me away from my desk for a deep breath.

Most of all, I am eternally thankful to my loving companion for life, Christine Hosey, who has had my back every day for 15 years. She spent countless night-walks helping me develop my ideas and how to present them, and countless day-walks recharging my batteries in the Fells or on the beaches of Chicago. She found my sanity when I couldn't, and she dusted it off. Christine has always reminded me of what's important. After all the other lessons, that was the one I needed most.

This dissertation is dedicated to my parents,

Karin and David McLure,

the seeds to my STEM.

And to my soul mates,

Christine and Hazel Hosey,

who would have loved me either way.

Contents

List of Figures

List of Tables

1 Introduction

Wouldn't it be nice if our machines could interact with us via sketching? A personal intelligent tutoring system capable of communicating with its user using sketching could collaborate on drawings of food webs to teach ecology, molecular bonds to teach chemistry, and free-body diagrams to teach physics. A personal design assistant could assist in exploring and debugging layouts by architects, urban planners, interior designers, landscapers, and user interface designers. A quarter of the world's population is illiterate. Five percent of the world's population is deaf and in developing countries, most of the deaf are illiterate. Algorithms capable of composing scenes based on descriptions and vice versa could support an ideographic babel fish. Sketching systems that use inspectable, verifiable spatial reasoning might even reduce human bias in solving important social problems that incorporate qualitative spatial constraints, e.g. drawing fair electoral districts (Bangia et al. 2017; Duchin 2018). More simply, many of us could use a digital assistant to help us organize and analyze our digital and physical notes. This work answers questions in pursuit of these sketching collaborators.

Collaborating on a sketch can involve difficult subtasks, including recognizing a person's ink as conceptual things, detecting intended spatial properties and relationships between those objects, interpreting conceptual properties and relationships from the spatial ones using task context, integrating visuo-spatial input with natural language input, performing domain-specific reasoning and social reasoning to determine how to respond or contribute to the content of a sketch in a way that adds value to the conversation, and, when appropriate, drawing original content. This dissertation focuses on the first of these subproblems – sketched object recognition – but the approach is intended to be compatible with solutions to the others.

Recognizing sketched input is a unique type of computer vision challenge. In some ways it is a simpler problem than recognizing photographs of real-world objects, which have rich variability in terms of light, color, perspective, and pose. Sketched objects often omit incidental visual effects like lighting to focus on the intended contours in a rendering and tend to be captured in a subset of their possible views and poses. On the other hand, details like lighting are potential clues for recognition. Furthermore, photographic images of objects in the real world are constrained by objects in the real world. A photo of a tiger is a 2-dimensional transformation of a 3-dimensional tiger, whose appearance is subject to the phenotypes in real-world tigers, as well as other regularities in how they appear such as the background environment (an undoctored photo of a tiger on the moon is hard to come by, but a sketch of a tiger on the moon can be generated relatively easily). Sketches are rarely photorealistic, and their departures from photorealism are often noisy and vary systematically with the artist. The fine-grained details are potential clues in a photograph but are more often distractors and sources of error in a sketch.

Furthermore, the space of concepts one might want to sketch is functionally infinite, and the most effective depiction of something may depend heavily on context. Since humans began drawing figures, thought to be at least 40,000 years ago (Aubert et al. 2018), our collection of drawn objects has been increasing. Many of the earliest known figurative illustrations were cave paintings of animals, typically painted from a profile view. The set of sketched concepts was modest and the mapping from label to targeted depiction was closer to one-to-one. When paper became widely available about 500 years ago, sketching became a common step in designing artifacts such as buildings (Goldschmidt 1991). Now humans use sketching to record a much wider variety of observations, but also to reason diagrammatically and communicate spatially. Sketching is taught as a skill in STEM disciplines to promote model-based learning (Quillin and

Thomas 2015). Designers use various types of sketches at different stages of the design process (Purcell and Gero 1998). There is now a many-to-many mapping between labels and our shared depictions (e.g. civil engineers' drawings of things labeled "water" are likely to differ systematically from chemists' drawings with the same label), and many depictions, especially in design, are deliberately novel. Depictions can vary systematically with the person drawing, so personalized software assistants that sketch might be more effective if they can learn personalized visual models of sketched objects. For example, a software collaborator might use these models to determine who is/was drawing what in a collaborative sketch, or simply to speak in a user's visual native tongue. This is all to say, there are an awful lot of sketched concepts out there, and an infinite number of sketch-able concepts. To keep up, our machines will have to be able to quickly learn from their human peers what novel sketched objects look like, and how they can be used to communicate and get things done. In other words, our AI should be able to learn to draw, and draw to learn, just like us.

Two key properties that should be prioritized in creating sketching collaborators are *flexibility* and *accessibility*.

By *flexibility* I mean two related things: (1) the ability to leverage learned knowledge across different contexts to make inferences without extensive training or hand-holding, and (2) the ability to support and integrate disparate types of reasoning (e.g. deduction, causal models, perception, analogy). These are critical properties of a useful collaborator, the first because of the scarcity of training data for specialized problems in the wild. One human's organizational, artistic or conversational style may differ considerably from another's while remaining internally consistent over time. Training data in the wild that is specific to interactions with one person or a small group is not going to naturally include millions, or even many thousands of data points –

at least, not before the software collaborator is expected to be useful. Making the problem worse, the style of content may be closely related to the specifics of whatever project or task is at hand. For example, an urban planner may work on projects from disparate regions, encountering differences in symbology and constraints. As task-specificity increases, naturally available training data decreases. Finally, sketching data as training data is relatively scarce because sketches are relatively expensive to produce - a picture is worth a thousand words, and it costs a lot of words in terms of the time and effort spent to create an example. This is all to say, big data techniques will only get us so far in creating personal sketching collaborators, as this challenge includes plenty of small (but rich) data problems.

The second aspect of flexibility is important because sketches are naturally multimodal, often accompanied by linguistic labels, cues, analogies and arguments. While training data from user-specific multimodal interactions may be scarce in terms of the number of examples, it is potentially rich with visual and linguistic structure. An important aspect of flexibility in sketch understanding is therefore to use representations expressive enough to integrate visual data with linguistic data, while bringing to bear social data, episodic memories, and semantic background knowledge. The representations should also accommodate different types of reasoning involving one or both of these modalities. Goldschmidt's work on the dialectics of sketching (Goldschmidt, 1991) documents an arsenal of sophisticated reasoning techniques verbalized by architects while developing designs for a library. The reasoning spanned recall and analogical projection (e.g. "I could see this as a casbah situation, where you have territories, confined territories…"), sketch generation and deictic reference (e.g. "…an atrium like this [drawn sketch]"), abduction (e.g. "Maybe there is no need [for an atrium], in which case…"), and deduction (e.g. "That [trees] means shadows."). The utterances were diverse and intertwined

with sketched content – some even verbalized while actively sketching – a concrete demonstration of why visual representations should be flexible enough to be combined with language representations and processed by reasoning systems.

By *explainability* I mean the ability of a software collaborator to inspect and explicitly communicate its reasons for its judgments, and an interface for precise tinkering, to an extent that allows for responsible human management. Whether it is a trading algorithm, a drone, a navigation system or a medical assistant, to monitor an AI and assess its reliability, we need to know, to a certain depth, why it makes the judgments it is makes. With modern artificial intelligence's sobering power, ubiquity and frailty, there is a moral imperative to attach expectations of explainability to our performance expectations. We should be able to ask our digital assistants "why", and we should be able to correct them in a precise way.

Predicate calculus is an expressive, highly structured, symbolic representation. The Cyc knowledge base, for example, is comprised of assertions, written in the CycL language, that store open-domain common-sense knowledge. CycL is a declarative ontology language that subsumes first-order logic but also includes higher-order quantifiers and modal operators. When grounded in a vetted knowledge base as in the case of CycL, predicate calculus is accessible to any knowledge engineer in a high-fidelity way. Even the designer of a neural network would be hard-pressed to communicate why the network made a particular decision. Moreover, in a cognitive architecture whose transactions are in predicate calculus, the problem of understanding and generating natural language about what has been learned is cleanly separated from how it was learned. One can independently focus on translating natural language to predicate calculus descriptions and back, in addition to reasoning about how to participate in a dialog, at which

point all descriptions of concepts and examples become much more accessible even to untrained users.

Analogy using structured, symbolic representations has had success as a technique for learning from relatively few examples (Chen et al. 2019; Liang and Forbus 2015; Wilson et al. 2016). A very prolific view of analogy from cognitive science is the Structure Mapping Theory (Gentner 1983), and its corresponding computational model, the Structure Mapping Engine (SME; Falkenhainer, Forbus, and Gentner 1989; K. D. Forbus, Ferguson, Lovett, & Gentner 2016) has been used in many cognitive models (Blass and Forbus 2016; Dehghani 2009; Friedman 2012; Kandaswamy 2017; Lockwood, Lovett, and Forbus 2008; Lovett and Forbus 2017; Lovett and Schultheis 2014) and AI tasks (Barbella and Forbus 2013; Chang and Forbus 2014; Halstead 2011; Klenk et al. 2011). In fact, Lovett's model of visual problem solving (A Lovett et al. 2009; Lovett and Forbus 2011; Lovett and Forbus 2017) was the genesis for most of the domain-independent visual-spatial properties and relations used here, and Kandaswamy's shape descriptors account for many of the rest. Structure mapping over predicate calculus representations has also been used to perform analogies involving multimodal data in order to answer textbook questions about biology and physics (Chang 2016). As discussed above, the ability to incorporate data from other modes of communication would be an important asset for a sketching collaborator.

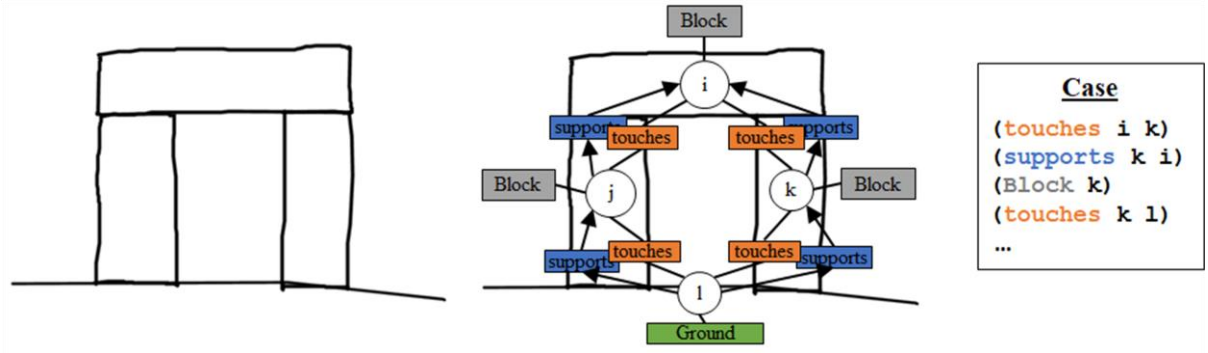


Figure 1.1: A sketch of an arch and a subset of its structured representation.

Structure mapping does not expect the input representations of an example to be from a fixed set designed specifically for the problem or class of problems, unlike feature-vector approaches. Instead, it expects them to be encapsulated in an arbitrarily sized set of assertions – or facts – called a *case*, representing the example. These facts ideally, but not necessarily, consist of predicates from a domain-general knowledge base e.g. the Cyc knowledge base. In Figure 1.1, the sketch of an arch (left) can be described in a case (right) that includes the relationships and attributes[1] that in the center are shown as rectangles overlaid on the arch, pointing to the perceptual entities (white circles) that they take as arguments. Any input that takes the form of a case can be labeled as an example of any concept. Structure mapping's expectations of its input add to its flexibility because using the same vocabulary of predicates to describe disparate concepts (even concepts learned from different modalities) facilitates more distant analogies.

[1] Note that for most datasets in this work we automatically encode ink using purely spatial predicates, as opposed to the partially conceptual relationships and attributes shown for the arch in Figure 1.1, which came from our only dataset whose encoding involved some manual assistance by a user.

These could allow a learner to generalize a classification rule beyond the task for which it was trained, or compose concepts into novel compound concepts, e.g. a *wheelbarrow of pumpkins*.

But why use a theory of comparison from cognitive psychology and visual representations from cognitive science as the basis for approaches for automated recognition of hand-drawn sketches? For one thing, it serves the goal of explainability. Comparisons are useful in teaching, argumentation, or for showing one's reasoning. A comparison meant for communicating with a human should look sensible to the human.

Another reason is that humans are very good at sketch recognition (Eitz, Hays, and Alexa 2012), while using reasoning that is flexible and accessible enough to bootstrap quickly in new domains and discuss why they understand something to be a member of a category. When human collaborators are our best working model of what we want from our automated sketching collaborators, imitation is a strategy worth pursuing.

A third reason is that by and large, the sketches we care about understanding are drawn by humans. The relevance of this relies on two assumptions: (1) A human sketcher typically draws with the intent to communicate a concept to other humans, and (2) the sketcher, presumably trying to communicate efficiently, draws what they deem to be the most important visual elements for recognition. Therefore, while sketches are more informationally sparse than say, a photograph, the information that remains in a human rendering is usually meant to be particularly amenable to humans' recognition processes. This assumption casts sketching more as a pragmatic communication device and less as an artistic expression, as that is the type of sketching we are more interested in endowing software collaborators with and the type that aligns with the datasets used in these experiments, which are described in Section 2.3.

A hallmark of an analogical approach to learning is an intermediate matching step. In the Structure Mapping view, this step is expected to (1) take unconstrained, potentially disparate, relational representations as input, (2) produce *one-to-one* correspondences, capturing which elements go with which elements across a group of these representations, and (3) produce a numerical similarity score between any pair of representations. These aspects underlie the potential of analogy to produce highly inspectable, communicable learned concepts, and to do so with relatively few training examples. The next two sections are meant to provide some intuition as to why. A structure-mapping approach to recognizing raw visual input does, however, necessitate an encoding step, by which symbolic relations are automatically generated to represent the visual input. Section 1.3 introduces a view of encoding and its implications. Sections 1.4 and 1.5 outline the claims and contributions of this thesis, and Section 1.6 provides a roadmap for what to expect in each of the chapters.

1.1 Palm Trees and Pineapples

Figure 1.2 shows four sketches of palm trees and one sketch of a pineapple from a large dataset of human-drawn everyday concepts (Eitz et al. 2012). Of course, the author and reader have likely seen visual representations of palm trees and pineapples before, but imagine, if these

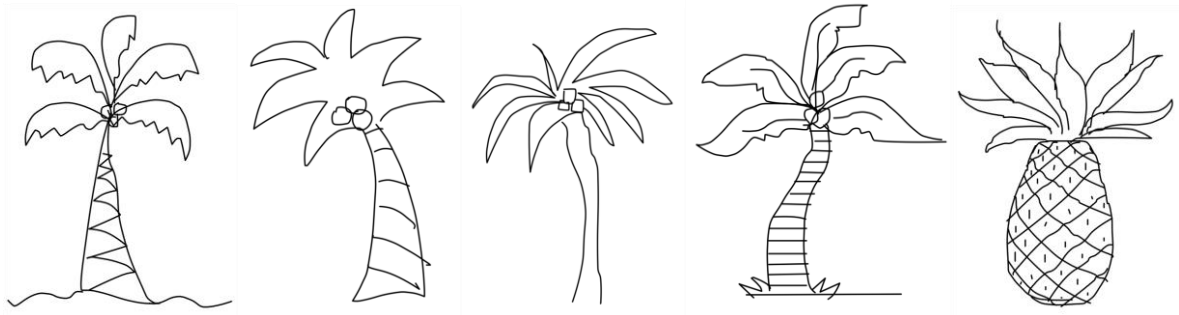


Figure 1.2: Four of palm trees and one of a pineapple (right) from (Eitz et al., 2012).

were the first examples you were seeing, what hints you may glean from them for how to recognize future palm trees and pineapples. First, palm trees and pineapples have some clear commonalities. In both, the large part on the bottom tends to be textured and has a vertical orientation. The group of similar oblong parts on the top radiate outward and come to points at acute angles. In the pineapple, the bottom part is convex and elliptical, whereas in the palm trees it tends to be more oblong, have corners, and include at least one concave, curved segment along its exterior contour. The texture of the bottom part is cross-hatched and dotted in the pineapple, but it is primarily made of horizontal lines in the palm trees. The palm trees also have some parts that the pineapple doesn't; they all have a group of small round parts near the center where the other parts come together, while half of them have a horizonal contour that overlaps with the bottom edge of the large bottom part and extends past it.

These visual indicators are communicable partly because they refer to discrete parts that the reader can identify. There are bottom-up clues in the ink that signal how to visually carve up a shape into parts. An important one is that we cleave shapes near their concavities (Hoffman and Richards, 1984), which is part of what allows us to see the top sections of the palm trees and pineapple as groups of parts, even when they are drawn as a single closed shape as in the second palm tree from the left.

Note that this separation into discrete parts using bottom-up information is not necessarily the most economical or discriminative way to carve up intermediate features for classifying this group of five examples. There could exist some extremely informative region or nonlinear function of pixels for which strongly opposite signals are detected in palm trees and pineapples, bypassing perceptual organization as an intermediate source of error, and discriminating all four examples more clearly and economically. This direct learning of complex but informative signals in the input is what makes deep neural networks such powerful classifiers. Indeed, the Sketch-a-Net convolutional neural network (Yu et al. 2015; Figure 1.4) reached super-human[2] performance in a recognition task on the same dataset (Eitz et al. 2012) from which the trees and pineapple in Figure 1.2 came. However, this direct, maximally discriminative end-to-end learning seems to come with risks that are only partially understood. Szegedy at al. (2013) presented adversarial examples – incorrectly classified images derived from correctly classified images, whose difference from the original is too slight a perturbation to even be perceptible. Figure 1.3 shows a correctly classified test example (left) and an

Figure 1.3: The original example on the left is imperceptibly different from the adversarial example on the right (Szegedy et al., 2013).

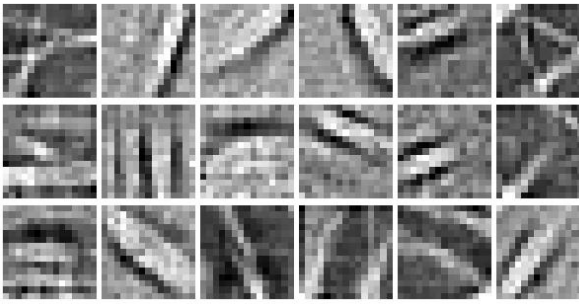


Figure 1.4: A random sample of the first-stage filters learned by the deep neural network Sketch-a-Net (Yu et al., 2015).

[2] To be precise, the Sketch-a-Net approach used stroke ordering information to divide input signals into channels. Humans performing the classification task had no access to this stroke ordering information.

incorrectly classified adversarial example based on it (right). They also presented an optimization procedure for reliably generating these adversarial examples. Most strikingly, they showed that a trained model could even be significantly confused by adversarial examples generated using a separate model, trained on different model parameters or even a different training set. Armed with some basic knowledge about the nature of a learned model's training procedure, a bad actor would not even need access to the model or the full details of its training to know how to trick it significantly.

Furthermore, the ability of an automated sketching collaborator to refer to and reason spatially with discrete, re-useable visual parts is a boon to explainability. For example, it can be useful for giving feedback to students on their sketched designs and worksheets (Forbus et al. 2018; Valentine et al. 2012; Wetzel 2014), especially if those parts can be labeled with concepts in a knowledge base, from which relevant spatial constraints can be gleaned. I further imagine this reasoning supporting novel sketch generation by composing new combinations of parts; asked to produce a palm tree with a cross-hatching texture, one might produce a prototypical palm tree example and cross-hatch within the confines of the part in the visual model of the palm tree that tends to be textured.

Finally, a recognition process that uses representations that refer to discrete, visually salient intermediate parts provides flexibility by resisting overfitting. Since humans have drawn the sketches, arriving at intermediate features more familiar to humans should mitigate brittleness.

Returning to the description of palm trees and pineapples above, note that it also refers to statistics about visual properties of the parts. Implicit in the communication that *the bottom part* is oblong and contains corners in the palm tree, whereas it is elliptical in the pineapple, presupposes that we have arrived at a shared visual model, or template, which contains a part –

the bottom part – and in each of the examples, we can know which ink to map onto that part in the model. In palm trees that part is oblong and has corners, in the pineapple it is elliptical. As Halstead points out, successive analogical matching of examples onto a shared template can explain how to track statistics of particular elements common to those examples, even when the input consists of arbitrarily complex relational representations, as opposed to a traditional feature vector (Halstead 2011). Feature-vector approaches do not require a matching step because the template is baked right into the training data as a set of key-value pairs. In the analogical case, the input does not have the keys baked in – in fact, its ability to operate on arbitrarily exotic relational representations is part of what makes it so flexible – but Halstead's key insight was that an assertion in an example case can still act like a feature in a vector as soon as the matcher determines which assertion (if any) it corresponds to in a sufficiently similar template that has been learned from sufficiently similar example cases. Structure Mapping outputs one-to-one correspondences between two representations that act like the keys, bridging the world of feature-vector learning techniques and the world of predicate calculus representations. For example, leveraging Halstead's template insight, Liang used structure mapping to make plausible inferences in a semantic web knowledge base by applying logistic regression over structured cases that were gathered using a breadth-first search through a knowledge graph. In this respect, analogical learning approaches are more closely aligned with feature vector-based learners than Inductive Logic Programming (ILP), a type of learning discussed in Section 6.1.2 that, like analogy, takes relational representations as input, but instead learns by searching for a first-order logic sentence (containing variables) that is satisfied by the positive examples of a category and not the negative. One such logic sentence in this scenario might be *if it has a circle, it is a palm tree.* This rule is simple to devise but is unlikely to remain discriminative when new concepts

come along. A more complex alternatives is *if it has a pair of parallel, vertically oriented curves below a cluster of circles, it is a sketch of palm tree*. The following complex rule is just as valid: *If all of its quadrilateral polygons are diamonds, it is a sketch of a pineapple.* It is hard to know which rules are more essential to palm trees, so ILP relies on heuristics to search for one. None of these rules captures a holistic view of the typical elements in a pineapple or these elements' typical properties, and they meant to remain valid when applied to all future examples. Matching parts in an example to those in a relational model to keep statistics over corresponding elements in the fashion of Halstead and Liang produces a more biased hypothesis space than searching for a logic sentence from scratch but is potentially less vulnerable to overfitting.

Finally, the reason we can compare the parts of the pineapple to the parts of the palm trees so readily is that they are visually similar. This is consistent with a classic result in cognitive psychology. People can more quickly tell that two stimuli are different when they are more different, but they can more quickly *identify* the differences when the stimuli are more similar (Sagi, Gentner, and Lovett 2012). We can readily communicate differences between palm trees and pineapples by referring to their corresponding parts because they are similar. In this analogical approach which uses Structure Mapping, the degree of structural similarity plays a key role in determining when to rely on correspondences between parts of examples to build statistics about these parts' properties and hypothesize classification criteria that refer to them. After all, when two objects are not very similar, how can it be decided *what goes with what* across them. Similarity is a gatekeeper for meaningful comparison.

1.2 Supervised learning of category models using structure mapping

In the structure-mapping view of supervised analogical learning adopted here, constructing probabilistic models for a category – say, *shark* – starts when the first example of the category is observed. The example, a structured representation comprised of relation and attribute elements, is stored in long term memory as a model, or *prototype*, of the category. Albeit a degenerative case, we can say that the original example is not only an example of *shark*, but also an example of our only existing prototype of a shark.

When a new positive example of *shark* is observed, it is compared to the existing prototype, and elements (relations, attributes and entities) in the new example are mapped onto elements those in the existing prototype according to the constraints of Structure Mapping Theory (Gentner 1983). This produces a numerical degree of similarity and a set of correspondences between elements. When the second example is sufficiently similar to the first, we can use their correspondences to turn the existing prototype into a generalized prototype. Elements that have an analog in both examples are seen as more characteristic of the prototype – they are deemed to have a high probability of being present in future examples of the prototype. Elements in either example that do not correspond to any element in the other are considered less frequent, or characteristic of the prototype. On the other hand, If the second example is not sufficiently similar to the stored prototype, it is stored as a new prototype of the category – a radically new type of shark, too different from the first prototype to have confidence in *what goes with what* across the two.

Future positive examples can be compared to the learned prototype(s), and if deemed to be sufficiently similar to one of them, they can be merged into it, with the probabilities of the

elements in the prototype updating to reflect the frequency with which the merged examples have contained an analog for the element. In this way, we can build up alternative structured, probabilistic prototypes that characterize various types of sharks. One way to determine if future unlabeled examples are sharks is by seeing how similar they are to one or more of our shark prototypes.

Of the elements that have been observed in all examples of a certain prototype, some may be critical to shark-ness. Perhaps a fish without a three-sided dorsal fin cannot qualify as a shark. These necessary conditions may be discovered by observing a *near-miss* (Winston 1970) – an example that is not a shark, but is nonetheless very similar to a shark that has been observed. Since it is not a shark but is the same as some shark(s) in almost every way, the differences between the examples can be seen as crossing a category boundary. One or more of the handful of differences may be necessary conditions for discriminating sharks from things that are nearly sharks. If a classifier only pays attention to similarity, it can be fooled by a future example that is a nearly a shark but fails to meet one of the shark criteria – e.g. it's missing a dorsal fin.

When a criterion is hypothesized for a prototype, it does not naturally extend to a second, disparate prototype for the same category, because it refers to elements grounded in the first prototype as opposed to variables that can be bound. How could we know which element in the second prototype it should refer to when we don't know what in the second prototype, if anything, corresponds to the referents in the first prototype? This inability to declare two elements equivalent across disparate representations is the same reason Halstead found statistics could not be reliably tracked across examples that are not similar (Halstead 2011).

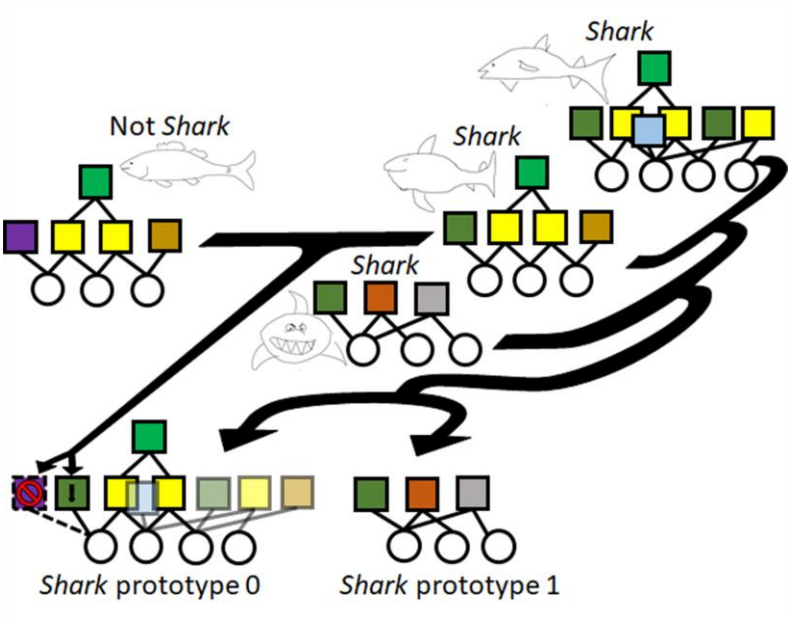


Figure 1.5: Structured representations of examples – three sharks and one non-shark – and prototypes that result from analogically clustering and comparing the examples.

Figure 1.5 shows the formation of multiple prototypes by clustering structured representations, and the formation of hypothesized criteria, which pertain to these prototypes, via a near-miss. Circles denote entities. Squares denote attributes and relations. Different colors denote different predicates. Translucent squares in the prototypes represent attributes and relations with frequency lower than 100% (50% in this case). The '**!**' marks a property that may be necessary and '⃠' marks a property that may be disqualifying.

Smith and Medin's seminal work on human category judgments (Smith and Medin 1981) suggests that humans have categories that have necessary conditions (e.g. triangle), categories whose membership is determined by similarity to probabilistic concepts (e.g. chair), and categories whose membership is determined by similarity to salient examples (e.g. suicidal patient). The perspective on supervised concept learning outlined here incorporates these three views of category membership. Salient examples can result from a lack of sufficiently similar

examples to generalize with (a single-example prototype for a category) and can coexist with probabilistic generalizations of the same category. Criteria can be learned that pertain to salient examples or probabilistic prototypes, as long as near-misses have been observed for them.

Another interesting property is that this approach can learn disjunctive concepts even if the hypothesized criteria are constrained to be conjunctions (an assumption that dramatically reduces the hypothesis space). The disjunction takes a form like:

> *The example must be similar to shark prototype A and there must be a part like the top triangular part in A that is above a part like the large center part in A,*
>
> *OR*
>
> *it must be similar to shark prototype B and the shape that is like the top part of B must have a vertical major axis.*

The ability to construct alternative prototypes of a category is crucial in sketched object recognition, where the drawer's style, the task-specific depiction conventions, and alternative perspectives (often not even projections of 3-D real-world objects) all affect how an object is sketched.

1.3 The Encoding Step

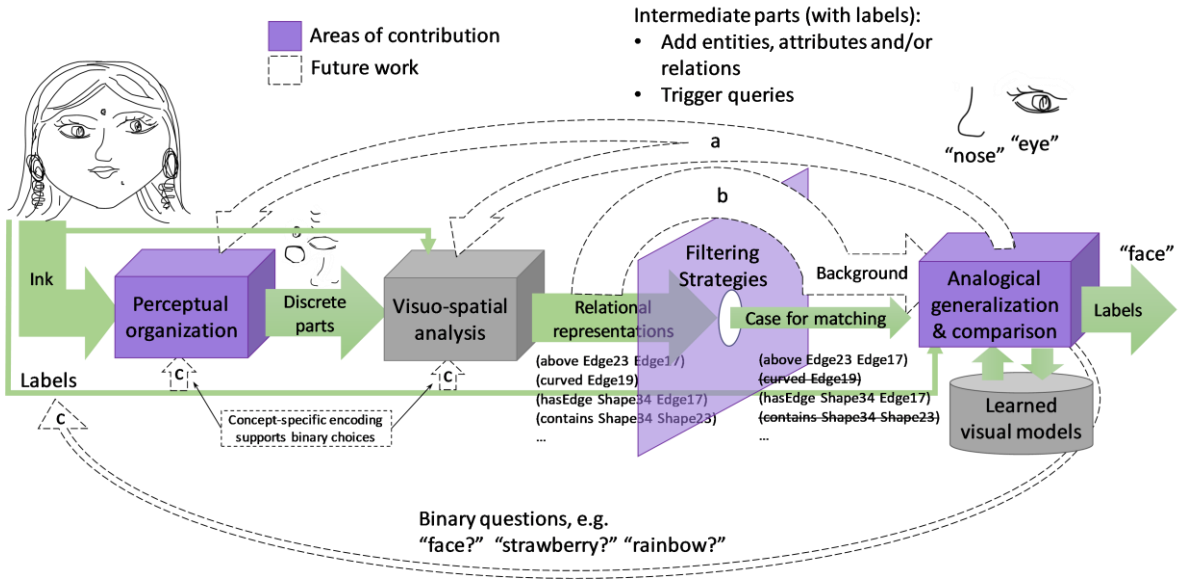


Figure 1.6: The information flow through this work's approach to bottom-up encoding and comparison (solid-colored components), and three possible extensions for future work (white, dashed arrows).

Structure mapping takes two structured representations as input. Using it to learn visual concepts by comparing raw visual instances requires an encoding step – a process by which a relational, qualitative, symbolic representation is generated to represent the raw, quantitative, non-symbolic visual input. For example, for any one of the palm trees in Figure 1.2, being able to feed analogical processes with the information that it has a group of small round closed shapes near the center (in a way that generalizes beyond this specific case) assumes, at least, visual processes for detecting closed shapes, assessing their roundness, and their relative position. Every input sketch used here is encoded with such visual processes before being fed to analogical processes for recognition. The non-white colored arrows in Figure 1.6 show our implementation of this pipeline in more detail. Encoding is broken down into three stages:

perceptual organization into a set of discrete entities, visual analysis for computing relations and attributes for those entities, and filtering for producing cases that can be tractably processed by structure mapping tools. Our contributions in this thesis are primarily to the stages in the pipeline in Figure 1.6 that are colored purple. The green arrows show the information that flows through our pipeline.

In Figure 1.6, the white, dashed arrows show threes ways (dashed arrows) that our approach could be extended in future work to (a) recognize intermediate parts that invoke rerepresentation, (b) incorporate the filtered-out representations as background for validate difference found by analogical matching, and (c) support concept-specific encoding to learn binary classifiers and answer binary questions, some of which may result from hypotheses about answers to open-ended questions with a more agnostic encoding.

Here we concern ourselves with the problem of organizing ink into visual primitives using bottom-up processes. Though we have not included any of the imagined feedback loops from Figure 1.6, note that the same visual processes and vocabulary that apply to primitives can apply to parts constructed from primitives (e.g. having recognized eyes, we can determine that the eyes are above the nose using the same process and vocabulary with which we determined that the edges representing eyelashes are above the circle representing a pupil). In this perspective, the only extension necessary to complete the feedback loop is detecting an object in a subset of a relational representation (versus recognizing the entire representation as an object) and determining which ink corresponds to it. I am not the first to imagine encoding this way; this is similar to the re-representational process at the core of Kandaswamy's cognitive model of forced choice tasks (Kandaswamy 2017). In that work, intermediate objects are detected using *interim generalizations*, which are in working memory from recent comparisons. This could be applied

to generalizations stored in long term memory, whose recall and application could depend on other top-down factors besides recency, including searching for a correspondence to bolster a comparison, or searching for known parts of the visual model for a label (e.g. *faces have eyes*), perhaps because this is a binary classification problem (e.g. *face or not face*) or because the label has been suggested as a possibility by some first-stage, superficial classifier (e.g. *it looks like a face or a strawberry or a rainbow*). The extraction of visual primitives from ink using bottom-up processes is still a key first step in any of these scenarios.

Even within a strictly bottom-up encoding process, there are open questions. Finlayson and Winston (2006) gather evidence for the idea that the ideal level of representation for analogical retrieval matching was an intermediate level, with enough detail for an informed and useful mapping, but not so much detail that the match is mired in them – the so called "Goldilocks Hypothesis." To generate useful representations bottom-up, we investigate strategies for minimizing verbosity and excessive detail while capturing the most relevant properties for recognition and enough structure to bias the matcher to choose productive mappings in a reliable way (reliable mappings, by our definition, are those that tend to preserve transitivity in a triad of three pair-wise comparisons between three similar cases – an idea we will return to in Section 5.1).

Chapter 3 deals with the encoding process. Specifically, we look at a sparser, more effective level of abstraction, derivable from edges, for carving sketched input into parts based largely on Saund's principles of tracing edge paths using tight closures and good continuity (Saund 2003). We go further by looking at strategies for putting hard limits on case size while prioritizing elements in a hierarchical structure that, together, cover as much of the ink as possible, before delving into details in a breadth-first fashion as space allows. We leverage the connective tissue

in our representations to be able to employ this filtering strategy at the knowledge level alone (without access to the sketch), and we show how it can be applied to multiple levels of perceptual organization in tandem to benefit from the most productive visual elements at both levels.

Next, we list the claims that this work provides evidence for, followed by its contributions. We conclude the chapter with a brief roadmap.

1.4 Claims

i. For sketched object recognition by analogy, cycles of edges, extracted based on closure and continuity, are a more effective and efficient level of qualitative representation than edges.

ii. Sketched object recognition by analogy can be improved by using a case filter on the input representations that greedily attends to the perceptual entities that cover the biggest portions of the least-represented ink.

iii. For sketched object recognition by analogy, with an entity filter based on ink/area coverage, attending to edge-cycles and edges together (with representational tissue connecting them) is more effective than attending to either one exclusively when controlling for maximum case size.

iv. Criteria learned from near-miss comparisons and refined by analogical generalization can improve the performance of a similarity-based classifier by censoring retrievals that violate a hypothesized criterion, but the benefit is not consistent when these near-misses are between analogical generalizations themselves – an approach with more plausible storage requirements.

v. Extending a similarity-based classifier with hypothesized criteria from near-misses can improve its performance when the hypotheses are generated in moderation, which is not the case when detecting near-misses between analogical generalizations at relatively low similarity thresholds.

vi. Using an ad-hoc analogical generalization at testing time, a linear support vector machine can be layered on top of similarity-based retrieval for higher classification accuracy than comparable similarity-based or near-miss-based classifiers, while requiring nothing beyond prototypes to be stored in long-term memory (the same storage requirements as a comparable similarity-based classifier) and being more robust to the compression of those prototypes.

1.5 Contributions

1. Edge-cycles, a perceptual organization method for ink, built on a planar edge graph embedding, that constructs shapes from cycles of the edges to produce a sparser and more effective representation for recognition.

2. A strategy for filtering visual cases that combines the strengths of edges and edge-cycles by progressively attending to visual entities that represent an outsized portion of the underrepresented ink. We provide a set of sufficient representational conditions for using this filter to weigh edges against edge-cycles in constructing hybrid cases, and for applying the filter at the knowledge level alone, obviating access to the external quantitative representation – the sketch – and, in theory, allowing it to be applied to generalizations or imagined examples.

3. A technique for detecting near-misses and hypothesizing criteria for category memberships from them, in conjunction with a principle for refining them via analogical generalization. These components are shown to potentially benefit classification in moderation, but also to potentially harm it if overdone.

4. The Analogical Boundary Learner (ABLe), an analogical approach to using support vector machines (SVM's) in conjunction with similarity-based retrieval and analogical generalization, in an SVM-KNN (Zhang et al. 2006) framework. ABLe augments similarity-based classification in a way that is more effective, more stable, more space-efficient, and more robust to compression of the training data (by analogical generalization) than prior near-miss-based discriminative approaches.

1.6 Thesis Roadmap

Chapter 2 provides the requisite background in the Structure Mapping view of analogy and the tools that implement it – the Structure Mapping Engine, a model of similarity-based retrieval called MAC/FAC (many are called, few are chosen), and SAGE, for the sequential analogical generalization of examples. It also goes into detail about CogSketch's processes for perceptual organization and its spatial representations. Finally, it describes the datasets used in the experiments to follow.

Chapter 3 looks at edge-cycles, a novel representation for supporting better, more tractable recognition. It also describes two alternative filtering strategies for putting top-down constraints on visual analogies and shows how to use the more effective of the two to coordinate edge- and edge-cycle representations for a hybrid scheme that is superior to each individually.

Chapter 4 looks at how to learn explicit criteria for category membership from near-misses, using structure mapping. Two near-miss approaches built on structure mapping tools are presented and tested on sketch recognition tasks, one of which uses a less scalable approach but demonstrates the value of near-misses for classification, and the other of which takes an approach that is more scalable but less clearly effective.

Chapter 5 explores an alternative method for leveraging near-misses using support vector machines. This technique achieves performance that is as good as or better than the better performing near-miss classifier, while being even more scalable than the more scalable near-miss classifier.

Chapter 6 reviews related work and Chapter 7 wraps up with conclusions and ideas for future work.

2 Background

2.1 Structure Mapping

2.1.1 Structure Mapping Theory

The Structure Mapping Theory (Gentner 1983) of analogy characterizes comparison and similarity judgments as a process of aligning structured representations. The matching process between two representations, base and target, is governed by several constraints: (1) *One-to-one correspondence* allows each element in the base to map to at most one element in the target and vice-versa (an injective mapping), (2) *Parallel connectivity* ensures that if two relations correspond, their arguments will correspond, and (3) *tiered identicality* only allows relations to correspond if they have identical predicates, or if aligning close predicates would support a larger relational match. The *systematicity bias* establishes a preference for larger, more interconnected aligned structures. In addition to correspondences, the matching process may

generate *candidate inferences*, which project unshared structure from the base onto the target and from target onto base, if and only if it is connected to the shared structure.

2.1.2 The Structure Mapping Engine (SME)

The Structure Mapping Engine (Forbus et al. 2016) is a computational model of Structure Mapping Theory. It matches two structured relational descriptions using a local-to-global

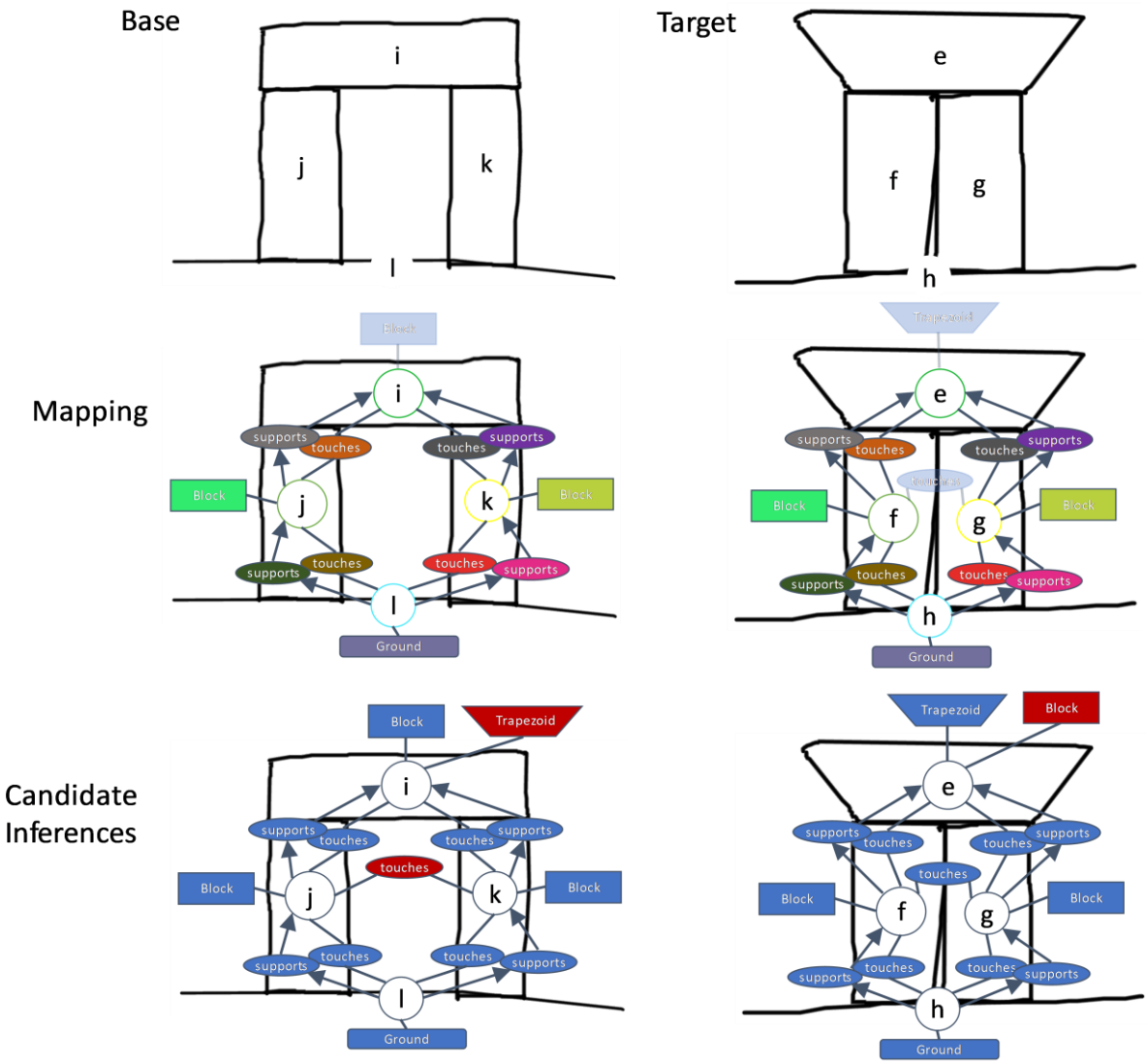


Figure 2.1: Two arches compared using SME. The middle row visualizes the mapping between their representations (actually a subset of their representations), and the bottom row highlights the resulting candidate inferences.

process. First, potential correspondences between identical predicates give rise to local match hypotheses, which then greedily coalesce into progressively larger, structurally consistent clusters. The result is a set of mappings - globally consistent sets of correspondences between the two descriptions. Each mapping contains a list of correspondences from entities and expressions in the base to entities and expressions in the target, color-coded in the middle row of Figure 2.1, a list of candidate inferences (from base to target) and reverse candidate inferences (from target to base), shown in red in the bottom row of Figure 2.1, and a structural similarity score. Mappings are scored using a trickle-down process that implements the systematicity bias by rewarding larger shared structure. In practice, similarity scores are often normalized to range from 0 to 1, by dividing by either the *self-score* of the base (the similarity score of its match with itself), the self-score of the target, or the mean of their self-scores.

Candidate inferences may contain *skolems*, hypothesized entities in the target which are projected from entities in the base that have no correspondence in the mapping. Because candidate inferences are only produced when they mention some shared (corresponding) structure, these skolems are only produced if the non-corresponding base entity participates in a relation with some corresponding entity.

SME is a heuristic procedure for finding mappings that optimize systematicity subject to the constraints of Structure Mapping Theory. This problem is similar to the maximum common induced subgraph problem for labeled graphs, which is NP-hard. SME's time complexity is $O(n^2 \log(n))$ for two equal-size cases of n items (entities and expressions) each.

2.1.3 MAC/FAC

MAC/FAC (Forbus, Gentner, and Law 1995) is a model of similarity-based retrieval built on SME. Given a probe case (a relational description) and a case library, MAC/FAC retrieves up to three similar cases from the case library. The algorithm has two stages, depicted in Figure 2.2. The first stage, MAC (many are called), computes fast, shallow similarity estimates between the probe and every case in the case library using dot products of content vectors. The content vector of a case stores a sparse encoding of the frequencies of all functors (predicates at the heads of assertions) in the case. For example, if the case includes 4 *contains* relations, the *contains* dimension of the case's content vector has the value 4. The content-vector dot-product takes the dot-product of two content vectors and normalizes the resulting scalar to range from 0 to 1 by dividing by the square root of the product of the magnitudes of the two content vectors.

The MAC stage allows the algorithm to scale to large case libraries. It passes the most similar cases (by default, at most three) to the second stage, FAC (few are chosen). FAC performs an SME comparison between each retrieved case and the probe in parallel. The

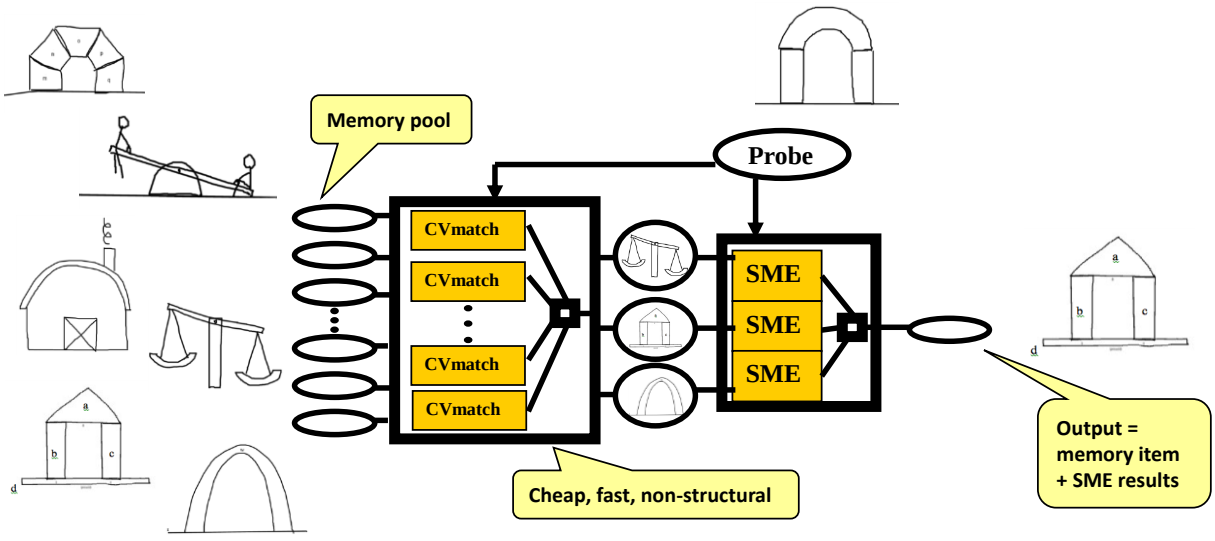


Figure 2.2: The MAC/FAC algorithm for similarity-based retrieval.

similarity scores returned by SME are used to sort the candidate retrievals and the most similar case is output (a reminding), plus up to two more if they are close to the top score.

MAC/FAC can be used to classify an unlabeled example by using it as a probe to retrieve similar examples from a case library of labeled training examples. The label(s) associated with the reminding(s) propagate to the unlabeled example. Taking the label of the top reminding approximates nearest neighbor classification with a structural similarity measure (it is an approximation because of the fast/cheap MAC stage). This classification scheme will be referred to in later sections as the *Retrieve-NN* approach. It will be used to evaluate encoding schemes in Experiment 2 (Section 3.5). Taking the top label from a weighted sum of the remindings (weighted by similarity score), approximates weighted K-nearest-neighbor classification. This scheme will be referred to below as *Retrieve-KNNW*.

2.1.4 Sequential Analogical Generalization Engine (SAGE)

The Sequential Analogical Generalization Engine (SAGE) is a model of analogical generalization and the evolutionary successor to SEQL (Kuehne et al. 2000). It has been used to explain progressive alignment, the human phenomenon in which abstractions form more readily from sequences of examples that differ incrementally. In SAGE, examples, in the form of cases (sets of propositions), are added sequentially to a *generalization pool*, where they are automatically clustered into probabilistic generalizations (cases themselves, where facts have a frequency associated with them) and ungeneralized examples (cases that were not sufficiently similar to any preexisting generalization or ungeneralized example). The SAGE algorithm consists of two stages: (1) *Select* at most one existing cluster to add an incoming example to, and (2) *Merge* the incoming example into the cluster.

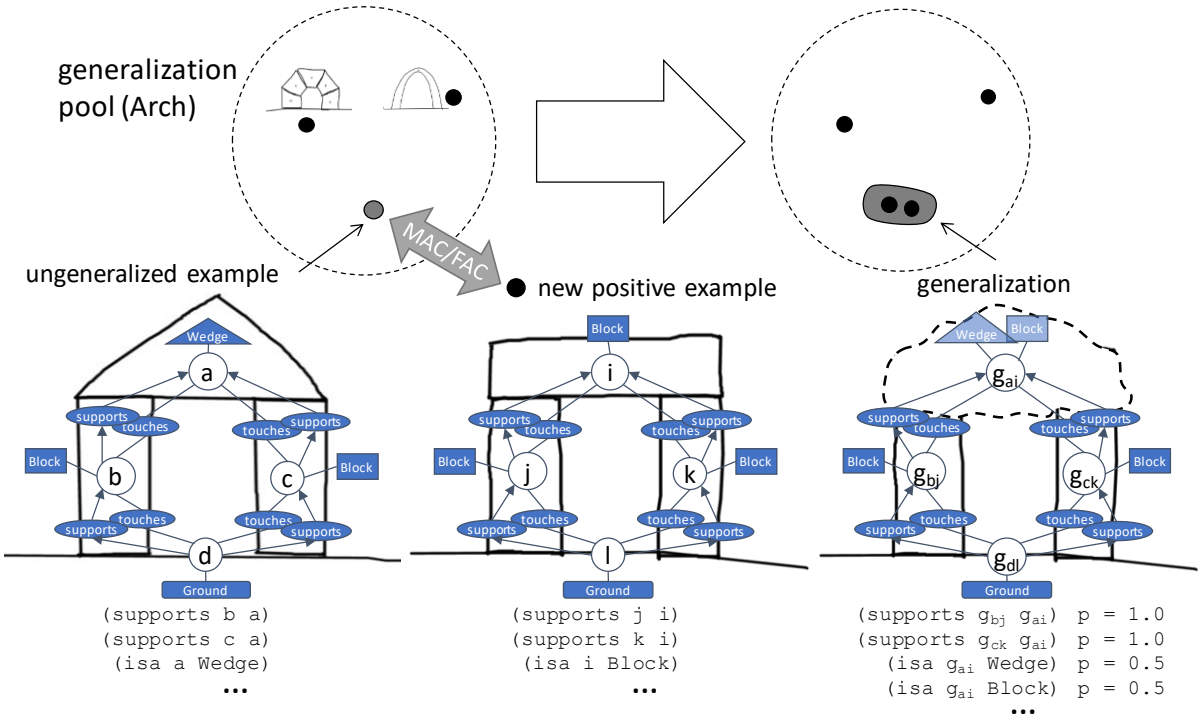


Figure 2.3: The formation of a probabilistic generalization by adding to a SAGE generalization pool a case that is sufficiently similar to a previously added, ungeneralized example.

Selection of a generalization partner is performed using MAC/FAC, treating the incoming example as the probe and the generalization pool as the case library for retrieval. The similarity score for each of the resulting remindings is normalized using the mean of the self-scores for the reminding and the probe. This normalized score is guaranteed to be between 0 and 1 since the mean of the two self-scores will be greater than or equal to the lesser of the two, and the similarity score between two cases cannot exceed the self-score of either case (the amount of relational structure on either side of a comparison are upper bounds on the amount of potential shared structure). The reminding with the highest normalized similarity score is selected to be merged with the incoming example if and only if its score exceeds the *assimilation threshold* of the generalization pool, which ranges from 0 to 1.

The select stage passes along the selected cluster to the merge stage, along with its analogical mapping with the incoming example. If the selected cluster is an ungeneralized example, as is the case in Figure 2.3, the incoming example will be merged with the ungeneralized example to create a new generalization. If the selected cluster is a generalization, the merge process will assimilate the incoming example into the generalization. If nothing is selected, nothing is merged, and the incoming example is added to the generalization pool as an ungeneralized example.

The merge stage begins by using the SME mapping returned by the select stage to create generalized entities from corresponding, non-identical entities in the base (incoming example) and target (selected cluster). These generalized entities will replace their corresponding original entities in the resulting generalization. When the target is a generalization, some entity correspondences may include a generalized entity, in which case that entity will be reused in the updated generalization. Using the mapping from base/target entities to generalized entities, SAGE translates each pair of corresponding expressions from the SME mapping into a generalized expression, adds this expression to the resulting generalization if it is not already there, and updates its associated probability. Expressions from the base and target that are not in the SME mapping are also translated, added to the generalization if necessary, and updated. The probability of an expression in the generalization reflects the proportion of its assimilated cases that contained an expression that corresponded to that (generalized) expression. For example, in Figure 2.3, the corresponding expressions end up with $p = 1.0$, whereas the non-corresponding expressions each end up with $p = 0.5$, since an analogous expression was found in half of the assimilated cases. If the next case to be assimilated were to contain an expression that mapped onto one of these, that generalized expression's probability would update to $p = 0.67$.

A generalization pool can be treated as a case library for MAC/FAC retrieval because SME can take a generalization as input for analogical matching just like any other case. The probabilities associated with expressions in the generalization act as weights on match hypotheses during the match process. Therefore, the higher probability structure has more influence on the analogical mapping. This means that an incoming example that shares some structure with the high-probability structure in a generalization, as well as a comparable amount of structure with low-probability structure in another generalization, will be more likely to be assimilated to the generalization with the higher-probability structural overlap. As a result, generalizations in SAGE tend to end up containing a wide distribution of probabilities. These probabilities separate characteristic structure (high probabilities) from noise (low-probability) across the generalized examples.

The approaches that will be described in Chapter 3 maintain separate generalization pools for every category label, so that similar examples will only merge if they share the same label. Positive examples of the category are sequentially added to its generalization pool, resulting in a set of generalizations for each category that reflect alternative patterns of characteristic structure for that category – below, these will sometimes be referred to as *prototypes*. Comparisons with SAGE generalizations of learned categories have been successfully used to classify unlabeled examples of musical genres (Dehghani and Lovett 2006), spatial prepositions (Lockwood et al. 2008), and sketched every-day objects (Lovett, Dehghani, & Forbus, 2007). McLure et al. (2010, 2011, 2012) describe a classification algorithm that uses similarity-based retrieval over the union of the generalization pools for all learned categories, which is a more scalable approach than iteratively comparing an unlabeled example to every generalization from every pool. In the same way, the *Retrieve-NN* and *Retrieve-KNNW* approaches described in the

previous section can be applied over a union of generalization pools. *Retrieve-NN* over generalization pools is used to evaluate encoding schemes in Experiment 1 (Section 3.4), and it is the foundation for the discriminative classifiers in Section 4 as well as the baseline used to evaluate them in Experiment 3 (Section 4.5). *Retrieve-KNNW* over generalization pools is used as the similarity-based baseline for evaluating a discriminative classifier in Experiment 4 (Section 5.3).

2.2 CogSketch, Perceptual Organization, and Spatial Representation

CogSketch is an open-domain sketch understanding system that has been used for sketch worksheets in the classroom (Forbus et al. 2017, 2018; Yin et al. 2010), cognitive modeling of spatial problem solving (Andrew Lovett et al. 2009) and preposition learning (Lockwood et al. 2008), tutoring in engineering design (Wetzel 2014) and other AI research. It provides an interface for drawing digital ink. Every pen stroke produces a polyline, a series of points. When utilities in CogSketch are used to import images of other formats, it ultimately produces polylines (a data structure native to the SVG format but requiring processing for bitmaps, e.g. for the MNIST dataset per Section 2.3). Sets of polylines can be manually grouped into conceptually meaningful objects called *glyphs*, either on-the-fly or post hoc, allowing users to manually segment ink into glyphs and then label the glyphs with concepts from the Cyc ontology. A multitude of qualitative spatial attributes and relations can be computed – some automatically, some on demand – about glyphs and groups of glyphs. Candidate visual-conceptual relations are made available based on the conceptual labels of glyphs, their spatial relationships, and the pose. For example, if glyphs A and B have each been labeled as

RigidObject, they touch without overlapping, and A is above B, then the On-Physical relationship is made available.

CogSketch provides routines for decomposing the ink of a glyph into discrete, visually salient parts at multiple levels of abstraction. This process is called *perceptual organization*. CogSketch can also compute spatial relationships (e.g. (above Edge1 Edge2)) and attributes (e.g. (isa Edge1 StraightEdge)) between the parts, much like at the glyph level. Aggregating a subset of these assertions can produce a structured, qualitative description of a glyph's visual parts, in the form of a case. Below we use the term *encoding* to refer to perceptual organization together with the computation and selection of qualitative descriptors.

The lowest level of granularity at which CogSketch can organize a glyph's ink is the *edge level*. Edges are generated by carving the ink into segments at intersections using the technique in Section 2.2.1 below, and at discontinuities in curvature using an adaptation of the Curvature Scale Space technique (Lovett et al. 2012; Mokhtarian and Suomela 1998). Attributes at the edge level describe things like relative length (e.g. longEdge), curvature, orientation (axis-alignment). Relations between edges can describe the nature of their junctions (e.g. acute, clockwise), relative orientation (e.g. parallel), overlap, and other relationships. See (A Lovett and Forbus 2011) for a full catalog of edge-level descriptors. Edges, by default, form an embedded planar graph, since junctions are placed such that they cover all locations where

Figure 2.4: A sketch of a rabbit from (Eitz et al. 2012), decomposed into edges by CogSketch.

two edges intersect. Figure 2.4 shows the edges (in different colors) and junctions (circles) for the edge graph embedding of a sketch of a rabbit.

Notice that this edge graph embedding covers the ink in a way that Is jointly exhaustive and pair-wise disjoint – every piece of ink is covered by exactly one edge or junction. This will be relevant for entity filtering in Section 3.3.2 below. However, CogSketch also supports the detection of *super edges*, compound edges that result from a sequence of edges connected end-to-end that together have relatively constant curvature (e.g. at a "T" junction). Super edges are described with the same edge-level vocabulary. The *sub edge* relation is added to relate super-edges to their sub-edges, making explicit the composition of the super edges.

Many of the purely spatial descriptors CogSketch can compute about glyphs come from Lovett's *shape level* representations (Lovett and Forbus 2011). This level has a richer vocabulary than the edge level. Some of its attributes and relations are the same as, or analogs of, attributes and relations at the edge level (e.g. `rightOf`, `parallelElements`, `largeSizeShape`). Others result from the additional variations that shapes can take beyond those of constant-curvature edges, such as containment, and various flavors of symmetry. There are still more attributes at the shape level that derive from properties shared by all the edges in the shape. For example, a straight shape is made up exclusively of straight edges, a convex shape is made entirely of convex corners between adjacent edges, and an axis-aligned shape is one whose edges are all axis-aligned.

Lovett's edge- and shape-level descriptors, in conjunction with the *group level* – a third level of spatial representation in CogSketch used for groups of shapes – formed the representational substrate for cognitive models of spatial problem solving, applied to Raven's Progressive

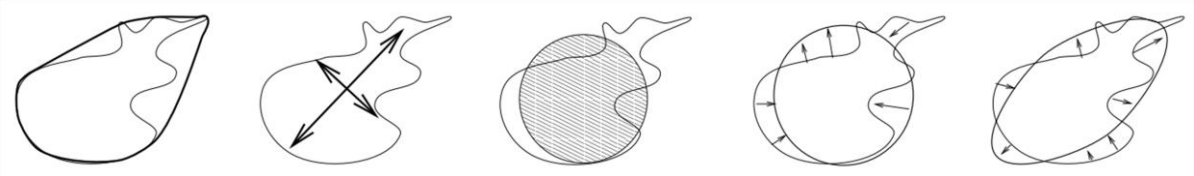


Figure 2.5: Shape descriptors implemented in CogSketch by (Kandaswamy 2017), which measure qualitative degrees along dimensions commonly used in object recognition, aggregated by (Peura and Iivarinen 1997).

Matrices (Lovett and Forbus 2017), geometric analogies (Lovett et al. 2009), and a visual oddity task (Lovett and Forbus 2011).

In addition to Lovett's shape-level of description, we use Kandaswamy's qualitative shape-level attributes (Kandaswamy 2017; Figure 2.5), implemented in CogSketch, which qualitatively characterize closed shapes along six quantitative shape measures commonly used in object recognition, five of which are aggregated by (Peura and Iivarinen 1997). Convexity is the ratio of the perimeter of a shape's convex hull to the perimeter of the shape itself. Solidity (added to the set by Kandaswamy) is the ratio of the area of the shape to the area of its convex hull. Eccentricity measures the ratio of the shape's principle axes. Compactness is the ratio of the shape's area to the area of the circle with the same perimeter. Circular variance is the normalized mean-squared error between the shape and its best-fit circle, and elliptic variance is the normalized mean-squared error between the shape and the ellipse with the same covariance matrix.

In CogSketch, the same level of representation can be applied to entities derived from disparate perceptual organization methods. Edges can trace ink, for example, or they can trace the skeleton of a closed shape via its Medial Axis Transform (Blum 1967; see Section 2.2.1).

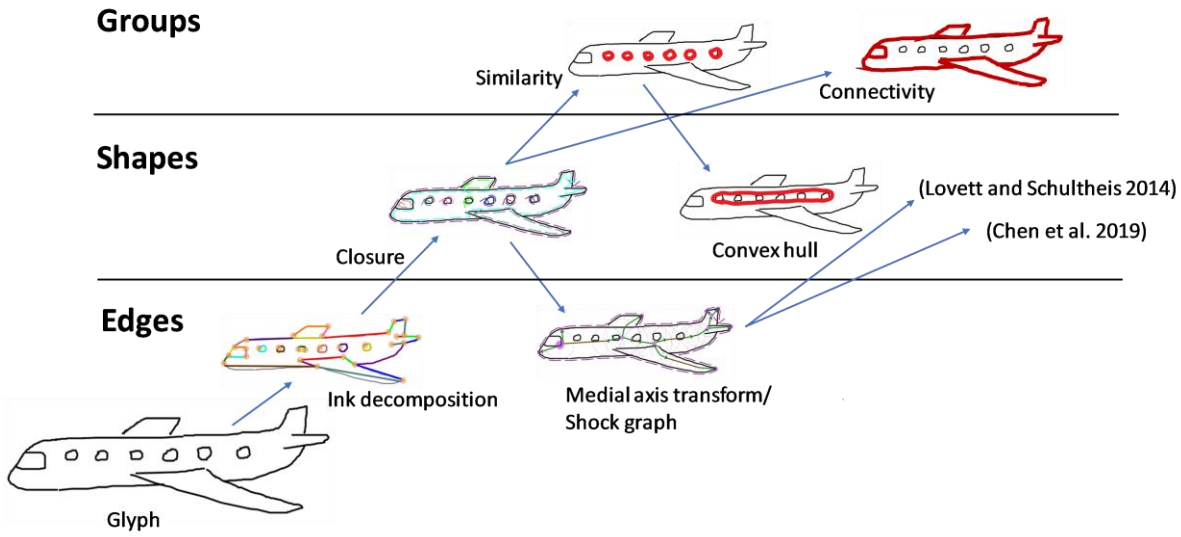


Figure 2.6: The levels of representation in CogSketch – edges, shapes, and groups – can each apply to many perceptual organization methods that can derive from other perceptual organization methods.

Shapes can describe glyphs, closed sequences of edges (McLure et al. 2011; see Section 3.1), convex hulls, or parts of a larger shape, e.g. derived from that shape's skeleton (Chen et al. 2019; Lovett and Schultheis 2014). These spatial transformations can be strung together, repeated, and used recursively. They can be conditional on the detection of qualitative properties (e.g. take the medial axes of all connected objects' curved perimeters). The resulting space of possible encoding schemes available for carving up visual input is rich and potentially overwhelming. But note some tractable aspects of this encoding scheme space. It results from a closed (but extensible) domain-general spatial vocabulary that spans three levels (Figure 2.6), and a closed (but extensible) set of well-defined, cognitively inspired, domain-general spatial transformations. It also provides a natural measure of complexity, since every scheme grows algorithmically from the same step – a decomposition of ink into edges. Complexity provides a natural preference if one were to search this space as part of a learning algorithm, an approach left for future work,

but nonetheless motivating the pursuit of spatial transformations and learning components in the present work.

Crucial overarching questions for structure-mapping based approaches to visual learning are: Which of these encoding schemes are effective? How should task demands and other top-down contextual signals factor into the choice of a scheme? How should top-down feedback from comparisons drive adjustments to the encoding scheme, and how should the adjustments be accounted for in learning? This work only addresses the first of these.

2.2.1 Decomposing ink into a planar edge graph embedding

The original decomposition of ink into edges is designed to result in a planar graph embedding, meaning it is drawn on a plane and no two edges intersect. The edges of the graph are curvilinear ink fragments and the nodes are junctions where multiple edges meet or where an edge terminates. Every edge connects exactly two junctions, and junctions can connect to zero or more edges. A zero-edge junction can result from a dot of ink. A many-edge junction would result at a place where many ink strokes are intended to converge, e.g. the center of a bike wheel with spokes. A two-edge junction would result from a discontinuity in curvature, per (Lovett et al. 2012).

This framing as an edge graph embedding contrasts with Saund's highly related work in tracing perceptual paths through curve fragments in the ink (Saund 2003). He starts with by creating a "junction graph", in which curve fragments instead serve as the nodes in the graph and pair-wise connections between ink fragments are the edges. Our term "edge graph" is meant to highlight this role reversal. Saund's junction graph is not necessarily planar, since many edges (nodes in the graph) that meet at a single intersection, like the center of a bike wheel's spokes,

may result in many pairwise junctions (edges in the graph) that cross one another to join pairs of edges.

Planarity in the edge graph embedding is enforced through the following constraints:

a) Every edge is a simple curve (no self-intersections).

b) No two edges intersect.

c) Edges can only intersect junctions at their endpoints.

d) Every edge endpoint must intersect the boundary of exactly one junction.

e) No two junctions overlap.

The CogSketch approach attempts to place junctions and edges in a way that abides by these constraints while best representing the *intended* geometry of the raw ink, meaning mending noisy discontinuities, such as gaps in ink or jitter in curvature. Given a set of strokes in the form of polylines, CogSketch immediately merges pairs of polylines whose end-points are very close together into longer polylines. Then it smooths and resamples them to reduce jitter and expedite later computations in a way whose trade-off between tractability and fidelity is parameterized (via the resampling interval). Then the strategy is to place junctions such that all the intersections and endpoints of these processed ink polylines are covered by some junction and no

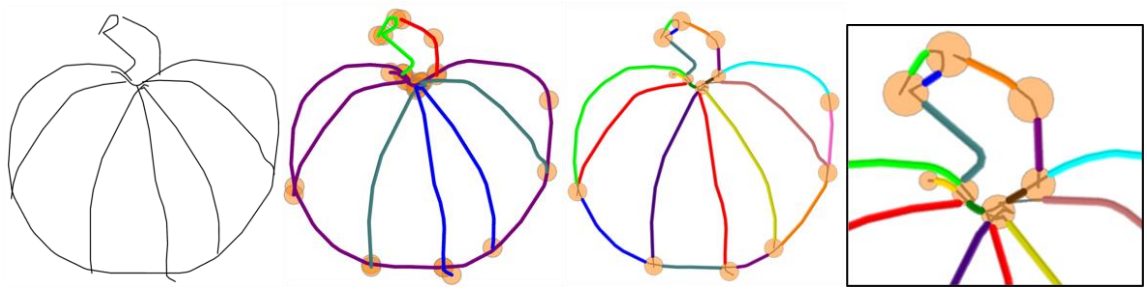


Figure 2.7: From left to right, (1) a sketch of a pumpkin from (Eitz et al. 2012), (2) its raw polylines and primitive junctions, (3) its final edges and junctions, and (4) a magnified view of an intricate region.

two junctions overlap, while preferring that nearby intersections/endpoints end up in the same junction and that junctions are not too large or too small (Figure 2.7). Then edges are extracted from each maximal ink segment that isn't covered by a junction. Every edge is associated with two directed edges, which traverse the edge in opposite directions.

Junction placement begins by finding all polyline intersections (self or mutual) and polyline endpoints in the processed ink. These points are marked as critical. CogSketch also finds corners and inflection points in the ink using a modification of the Curvature Scale Space (CSS) corner detector (Lovett et al. 2012; Mohktarian and Suomela 1998) and marks these points non-critical. For every critical and noncritical point, a new junction (a circle) with a default size proportional to the size of the glyph is centered on the point. These primitive junctions are allowed to overlap. Clusters of overlapping junctions are identified, and the junctions in each cluster are collectively modified such that all critical points overlapping the cluster are still covered by one of the resulting junctions, but none of the resulting junctions overlap. Specifically, we resolve the cluster by resorting to the following alternatives in order, resetting if and when each fails, until the last resort, which is guaranteed to succeed:

1. Find a location at which a junction of the default size can cover all critical and non-critical points in the cluster. If found, replace the cluster with that junction.
2. Shrink all junctions in the cluster as much as necessary to eliminate overlap without going below the minimum junction size.
3. Find a location at which a minimally sized junction, up to the maximum junction size, can cover all critical and non-critical points in the cluster. If found, replace the cluster with that junction.
4. Discard the junctions corresponding to non-critical points.

5. Find a location at which a minimally sized junction *of any size* can cover all critical and non-critical points in the cluster. Replace the cluster with that junction.

Our minimum junction diameter is the width of the narrowest piece of ink in the junction, and the maximum junction size is twice the default junction size.

Edges are extracted by subtracting the junctions from the processed ink. Every resulting polyline becomes an edge. Every edge connects two junctions, and no two edges intersect, since all intersections were covered by the junctions. The location where an edge meets the border of a junction is called a *boundary point*. Each boundary point stores a junction-referenced orientation, denoting the orientation of the ray pointing from the center of the junction to the boundary point, as well as a local edge orientation, denoting the local orientation of edge as it leaves the boundary point. These properties are relevant to the procedures for tracing edge-cycles in Section 3.1 below.

2.2.2 The Medial Axis Transform and Shock Graphs

The medial-axis transform (Blum 1967) of a region is the set of points on the interior that have more than one closest point on the exterior. This representation has been popular in computer vision for its ability to capture a shape's qualitative properties in a way that relatively stable with respect to articulations and viewing perspectives (Siddiqi and Pizer 2008). The medial axis transform is especially well suited for organic objects or figures, since it traces a shape's skeleton. There is evidence from cognitive psychology that the human visual system computes skeletons for shapes in general as an early visual process (Firestone and Scholl 2014), and that those shape skeletons play an outsized role in similarity/discrimination judgments compared to surface level properties (Lowet, Firestone, and Scholl 2018).

CogSketch can compute the medial axis transform given a cycle of edges as the exterior and any number of cycles of edges as interiors, provided the contours of the edge-cycles don't intersect.

There is an issue with using the medial axis transform on a region with hand-drawn contours. They grow hair. This is because jitter in the ink produces minor unintended convexities along the exterior. We employ a technique inspired by Bai et al. 2006), which uses the exterior edge segmentation to prune the medial axis transform. They prune any medial axis point whose *generating points* – its closet points on the exterior – lie on the same edge. Insofar as the exterior edge segmentation ignores unintended convexities, the medial axis will omit their corresponding branches, as in Figure 2.9. In our case, we obviate the pruning step by using the edges as wave-fronts in the grassfire algorithm that computes the medial axis transform.

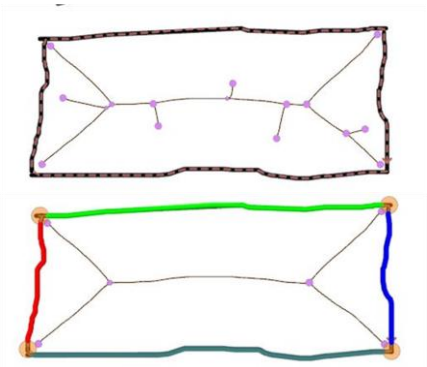


Figure 2.9: (Top) An approximate medial axis transform with hair resulting from minor, unintended concavities in the ink. (Bottom) A medial axis transform that does not include points whose generating points lie on the same exterior edge.

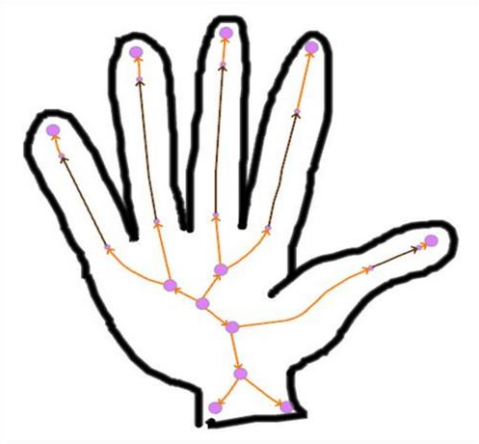


Figure 2.9: The shock graph of a sketched hand. The arrows point toward decreasing radius function. The brown edges are segments of relatively constant radius function.

CogSketch can compute a *shock graph* (Siddiqi et al. 1999) from a medial axis transform. The shock graph, an edge graph embedding, segments the medial axis at points of qualitative changes in the *radius function* – the distance from a medial axis point to its generating points. The resulting edges are described

with CogSketch's edge-level vocabulary. They are given additional attributes characterizing their radius function – whether it is decreasing or constant, its magnitude relative to the rest of the skeleton (wide/medium/narrow/very narrow), and the angle at which its generating points converge (acute/obtuse/right). They participate in additional relationships that capture changes in radius function between neighboring edges. For example, the pairwise combinations of the three edges at the center of the hand in Figure 2.9 are each connected by a source junction relationship.

We employ shock graphs in two of our datasets (explained in Section 2.3). In both cases, we prune the edges from the shock graph that are terminal (disconnected at one end) and have decreasing radius function. These are the outmost edges that reach into the corners of a shape. In the hand in Figure 2.9, eliminating these edges would remove the pair of edges reaching from the center of the wrist to its corners, as well as the edges in the fingertips.

2.3 Sketch Datasets

This section describes the datasets used in the experiments below. All involve hand-drawn input to some extent, and all ultimately ended up in the form of polylines in CogSketch, where a qualitative representation was encoded.

2.3.1 Berlin25

The set of everyday sketched objects from TU Berlin (Eitz et al. 2012) is, to the author's knowledge, the largest corpus of hand-drawn sketched objects (as opposed to symbols – see Section 6.2). Amazon's Mechanical Turk was used to collect 20,000 sketches of 250 different categories (80 per category) from 1,350 unique participants. The categories include various animals, plants, household objects, vehicles, musical instruments, toys, body parts, clothing,

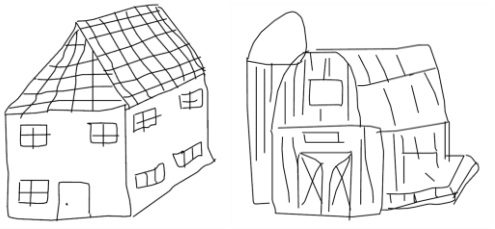


Figure 2.10: Two examples from our subset of the TU Berlin dataset – a house and a barn – that contain textures.

food, weapons, tech gadgets, celestial bodies and buildings, as well as some non-real-world categories such as angel, dragon, mermaid, and flying saucer. The median drawing time for each instance was 86 seconds and the median number of strokes was 13.

In the experiments here we used a subset of 25 of the 250 categories because our attempts to run on the full set were stymied by scalability issues. Our database access operations became prohibitively slow as the knowledge base grew to contain more than several million assertions, which wasn't enough to store the full representations of the full set of categories (the edge-based representations for all 250 categories took about 20 million assertions, let alone the hybrid edge/edge-cycle representations introduced in 3.2). A primary reason for the size of the representations were objects drawn with textures, which result in regions densely filled with small repeating visual structures (e.g. the grid of tiny quadrilaterals created by cross-hatching in Figure 2.10). The encoding schemes used here compute many relations over all pairs of neighboring atomic entities (edges or closed shapes), thus textures result in serious representational bloat. The implications of this issue are further discussed in Section 3.6.

In the interest of accruing more datapoints for comparing the approaches tested, a subset of 10% of the categories was chosen (once, up front, before seeing any per-concept results) to focus on simpler concepts that would not contain textures. These 25 categories are shown in Table 1. In Figure 2.11, a scatterplot shows the relative complexity of the Berlin25 subset in terms of the average number of entities per case, and the average number of facts per case. Averaged over all categories, the edge representations contained 1070 facts and 30 entities per case in the full dataset and 615 facts and 20 entities per case in the Berlin25 subset. The edge-cycle representations discussed in Section 3 below went from 490 facts and 12 entities per case in the full dataset to 232 facts and 7 entities per case in the Berlin25 subset. To the author's surprise, two of the concepts chosen for the Berlin25 subset, House and Barn, still often contained textures (Figure 2.10). These concepts show up as outliers in the scatter plot for the Berlin25 edge-cycle representations (purple).

Table 1: Categories in Berlin25, our subset of (Eitz et al. 2012).

Apple	Banana	Barn	Bowl	Cloud
Cup	Envelope	Eyeglasses	Fish	Foot
Frying Pan	Hammer	Hand	Hourglass	House
Knife	Moon	Mug	Mushroom	Nose
Skull	Spoon	Tent	Tooth	Wine Bottle

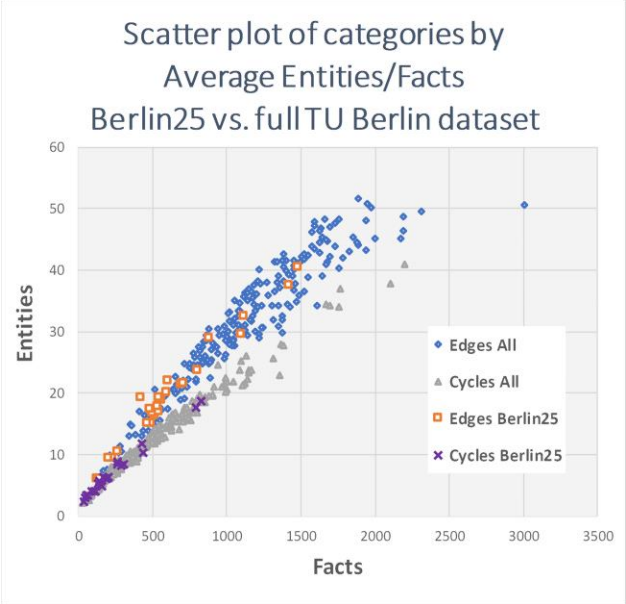


Figure 2.11: A scatter plot of the categories in the TU Berlin dataset and Berlin25 (the subset used here), plotted according to the average number of entities in the category's cases (x), and the average number of facts in the category's cases (y). The edge-based and edge-cycle-based versions of the cases are plotted (edge-cycles are discussed in Chapter 3 below).

2.3.2 Freeciv Geography

Freeciv is an open-source strategy game that takes place in a rich 2D environment. The game interfaces with CogSketch and Companions (McLure and Forbus 2012), which allows for spatial computations over geographical *blobs* – regions of tiles of a constant type (e.g. land, water). It also allows a user to draw on the Freeciv map. The Freeciv Geography dataset consists of 60 examples spread evenly over 6 geographical concepts: isthmus, strait, bay, peninsula, island, and archipelago (examples shown in Figure 2.12, left). Each instance is a user-drawn closed shape overlaid on a Freeciv map, enclosing a region.

An instance was encoded by (1) zooming to the user-drawn glyph, (2) computing pruned shock graphs for the visible blobs, and (3) computing qualitative representations to describe the edges in the shock graph, as well as the topological relationships between the user's glyph and the shock graph edges, specifically containment and intersection. Figure 2.12 (right) shows the shock graph edges and the user glyph in an instance of *Strait*.

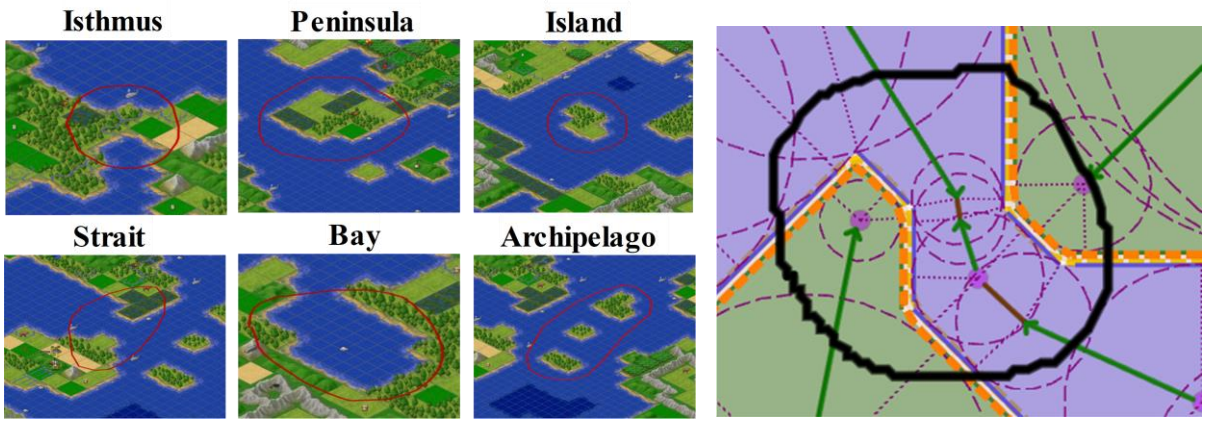


Figure 2.12: (Left) One stimulus from each of the six concepts in the Freeciv Geography dataset. (Right) The perceptual organization of the encoding scheme used. A pruned medial axis transform was computed for terrain/water blobs, from which shock graphs were computed.

This dataset only tangentially qualifies as hand-drawn, but still serves to demonstrate and compare the efficacy of learning approaches for visual input.

2.3.3 Sketched Concepts 2010

This dataset was first used in (McLure et al. 2010). It contained 44 examples spanning 6 categories in addition to the *null* label – i.e. it contained instances that were not examples of any category. The categories are shown in Table 2 and some examples are shown in Figure 2.13. Each example was drawn by a person in CogSketch, using multiple glyphs. This is the only dataset in which the segmentation of ink into entities was not automatic. Basic spatial relationships (positional relationships and topological relationships) were computed between glyphs, and conceptual labels from the Cyc ontology (e.g. Bone-BodyPart) were added to the glyphs. Visuo-conceptual relationships were then computed between glyphs in a semi-automated way, based on the spatial relationships and knowledge about the conceptual

Table 2: Categories in the Sketched Concepts 2010 dataset.

Arches:	8	Bridges:	4	*False arches:*	8
Skeletal arms:	4	Triangles:	4	*False triangles:*	4
Skeletal legs:	4	Quadrilaterals:	4	*False quads:*	4

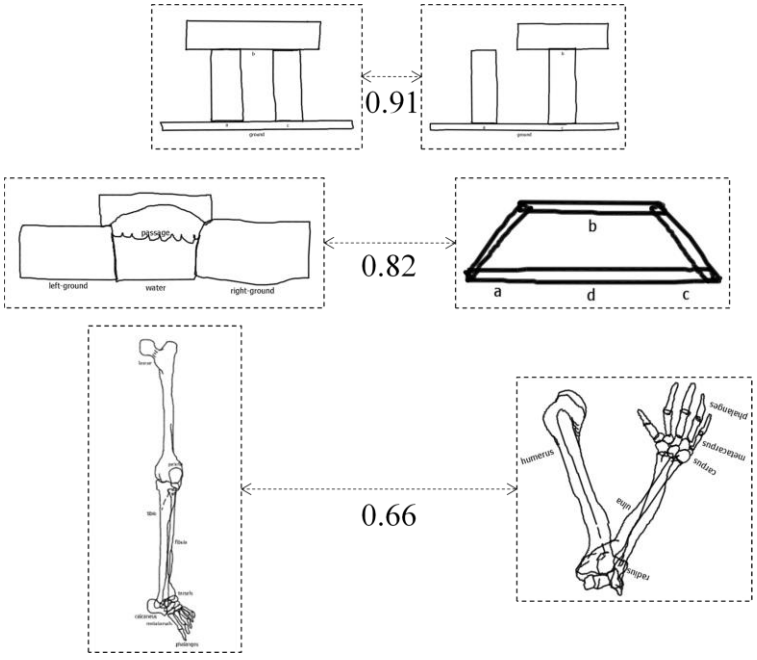


Figure 2.13: Examples from the Sketched Concepts 2010 dataset, with normalized similarity scores shown.

label. For example, the On-Physical relationship is manually selected from a list of automatically generated candidate relations that can hold between two rigid objects that are touching, one above the other.

The sketches in the Sketched Concepts 2010 dataset were drawn by two of the authors of (McLure et al. 2010), where each category was drawn entirely by one of the two.

2.3.4 Sun Up to Sun Down

This dataset was first published in (Lovett et al. 2006). It consists of 72 examples spread evenly over 8 categories of everyday objects, exemplified in Figure 2.14. These objects were drawn by 9 different participants – one sketch per category per person. The participants were shown the same stimulus for each category – images from the book Sun Up to Sun Down (Buckley 1979), which illustrates physical processes with simple drawings. The stimuli were provided to encourage the participants to draw from the same perspective. They were asked to only include what they thought was necessary to communicate the concept.

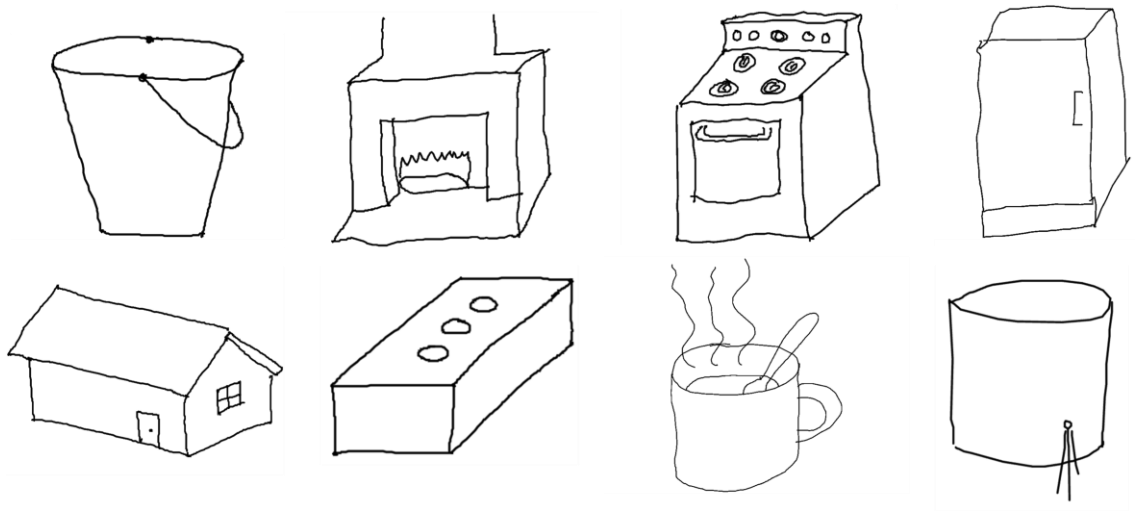


Figure 2.14: One sketch from each category in the Sun Up to Sun Down dataset: (top) Bucket, Fireplace, Oven, Refrigerator, (bottom) House, Brick, Cup, Cylinder.

In this dataset every sketch was a single glyph, so the CogSketch's perceptual organization routines were responsible for carving the sketch into entities.

2.3.5 MNIST

The MNIST dataset (LeCun et al. 1998) consists of 70,000 hand-drawn digits split into a training set of 60,000 and a testing set of 10,000. The images are in bitmap format, 20-by-20 pixels centered on a 28-by-28 pixel field. We used a version converted to Cogsketch's polyline format by Kezhen Chen and Irina Rabkina (Chen et al. 2019). This conversion process produced a glyph for each digit in which the polylines formed a boundary around the original ink, rather than tracing along the ink strokes themselves. To recover a more intuitive decomposition in

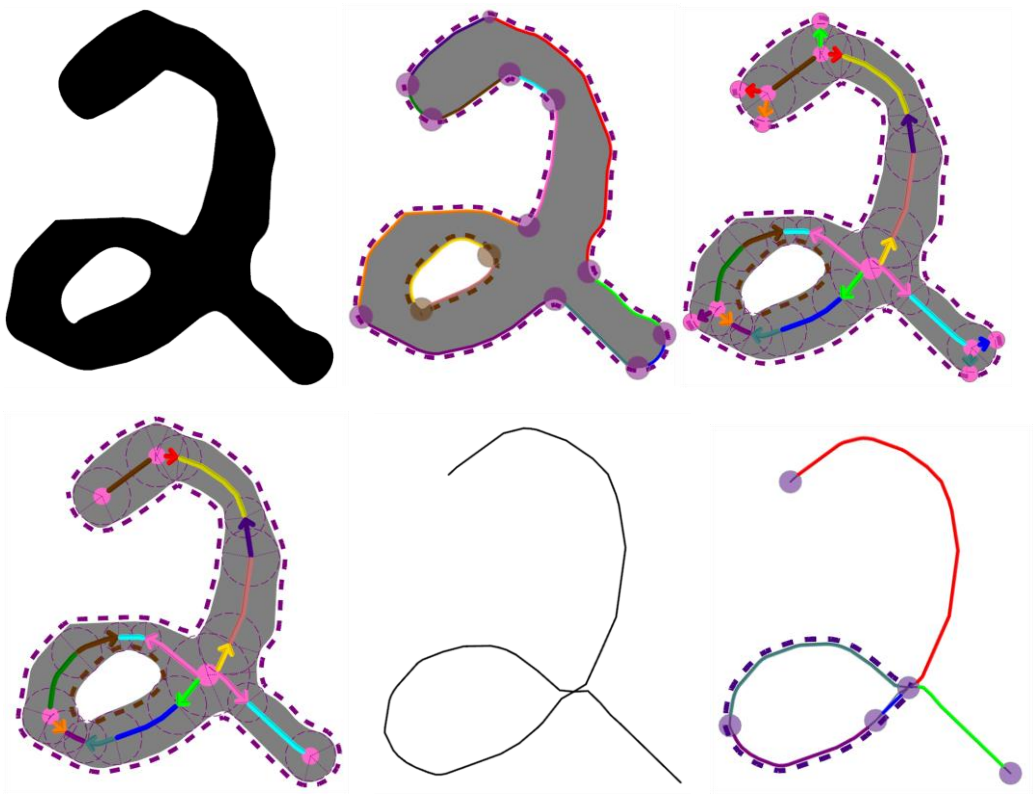


Figure 2.15: Stages of extracting a skeleton decomposition for an MNIST digit.

which edges represent segments of ink, we used CogSketch's (pruned) shock graph perceptual organization routine to extract the skeleton (Figure 2.15). The edges in the pruned shock graph were treated as ink polylines and decomposed into edges using the standard method for the ink in a glyph (Section 2.2.1), thereby losing the radius-function-based segmentations of the shock graph.

When using the MNIST dataset below, we selected training sets randomly from the MNIST training set (subsets ranging from 10 to 500 examples), and testing sets were randomly selected subsets of 500 from the MNIST testing set.

2.4 Companions and Experimentation

Companions (Forbus, Klenk, and Hinrichs 2009) is an agent-based cognitive architecture with a ubiquitous role for analogy. It has been used for various tasks such as solving physics problems (Klenk and Forbus 2009), learning by demonstration (Hinrichs and Forbus 2012), and extracting qualitative process model fragments from text (McFate, Forbus, and Hinrichs 2013). A Companion consists of several types of specialized agents. The *session reasoner* is the hub for domain-specific reasoning. A *tickler* is an agent that specializes in long-term memory operations such as similarity-based retrieval and analogical generalization. There are agent types that deal with specific interaction modalities. For example, a spatial agent interfaces with CogSketch. Through the CogSketch agent and a set of individual sketch agents that it manages, the Companion has access to the full CogSketch API including its catalog of spatial reasoning capabilities and (shared) control over many of the UI operations. It can organize a sketch according to any available perceptual organization strategy and attend to any subset of the available attributes and relations. Finally, the Executive is responsible for monitoring and

managing the other agents. The Executive can spawn worker agents to delegate parallelize-able workloads, and the workers spawn their own ticklers of spatial agents as necessary.

The experiments in this work were run on a Companion. The session reasoner decomposed an experiment definition into a set of conditions to test, and conditions into a number of independent trials (K trials for K-fold cross-validation). The session reasoner forwarded along these trials to the executive to delegate, where they waited for an available worker to take them up. Workers sent trial results back to the session reasoner for compilation and analysis. The error bars in all the results presented below are based on the standard error across these trials.

The apparatus was designed for longevity. Workers periodically performed self-maintenance (e.g. compressing their knowledge base, clearing working memory). The executive also performed maintenance by automatically rebooting any agent's lisp process that became unresponsive or whose heap became too bloated, first allowing the agent a chance to record its state in its knowledge base to pick up where it left off. The largest experiments ran for the better part of a week, spawning hundreds of workers on a single Companion that ran up to 57 agents (several hundred threads) at a time across 20 nodes in our cluster.

2.5 Summary

To apply structure mapping to hand-drawn input for recognition, the ink must be encoded into an intermediate qualitative, relational spatial representation. Encoding involves perceptual organization, the carving of ink into discrete entities, and a set of queries for detecting – and outputting assertions for – qualitative attributes of those entities and relationships between them. Section 2.2 provides a primer on how this process unfolds in CogSketch. It explains the foundational edge-based perceptual organization scheme in CogSketch as well as the edge-based

assertions that encode the arrangement of the resulting entities. It also provides details about the shape level of representation in CogSketch.

Choosing an encoding scheme is nontrivial because if the set of encoded properties is inappropriate, structure mapping may be bogged down by excessive detail, distracted by irrelevant details, or deprived of relevant details for learning a visual concept. The next chapter describes and evaluates an alternative perceptual organization scheme, as well as strategies for mixing and filtering representations at the edge and shape levels of representation. The goal is to develop more effective and tractable encoding schemes for learning sketched concepts by analogy.

3 Encoding Digital Ink for Tractable Analogical Learning

In this chapter, we describe edge-cycles, a shape-level method of perceptual organization derived from edges, for encoding more abstract visual entities to represent sketched input more concisely. We evaluate how edge-cycles compare to edges for analogy-based sketched object recognition, as well as the value of combining the two into a hybrid representation. In the interest of tractable learning, we also explore filtering strategies for encoding structured representations of fixed size and examine their efficacy.

There are important top-down clues to how to parse ink into edges and shapes. Recognizing that an object is an upright, open-top container may lead to an analysis of the shapes of the closures in the sketch to decide whether it is a cup, a bowl, a mug or a bucket. However here we only evaluate bottom-up strategies for perceptual organization, meant to precede these more sophisticated analyses.

3.1 Edge Cycles

Edge cycles are circular lists of contiguous edges that form a closed path in the edge graph described in Section 2.2.1, i.e. a sequence that ends where it began without traversing the same edge twice in the same direction. Edge cycles are limited to cycles in the graph that do not self-intersect, so that they form curvilinear polygons and are therefore describable using the shape vocabulary in CogSketch.

Saund identified two types of perceptually salient closed figures that appear in sketches – those that tightly circumscribe empty regions, and those that follow smooth, continuous contours (Saund 2003). He therefore searched for figural closures using two sets of tracing preferences, maximally turning preferences and smooth continuation preferences. We adopt these sets of

preferences to find edge-cycles using three separate tracing algorithms – one following maximal turns to find tight regions, one following the best continuation to find the smoothest contours, and one that mixes the two, following continuation when it is sufficiently good (such that the edges on either side could be seen as one continuous edge) and maximal turns otherwise.

3.1.1 Atomic and perimeter edge-cycles

A maximally turning tracing method will trace around the faces in the planar edge graph embedding introduced in Section 2.2.1, i.e. the regions of whitespace. One of the faces in the edge graph embedding will always be the background (infinite) whitespace. This face is traced by the perimeters of the outermost *edge-connected objects* – a term for the connected components in the edge graph embedding, i.e. maximal sets of connected junctions and edges. An edge-cycle is instantiated for each edge-connected object's perimeter. These are called *perimeter edge-cycles*. In Figure 3.2(a), the spout and the cylinder are different edge-connected objects. In Figure 3.2(b), the spout and the cylinder are connected by the stream of fluid, and therefore are part of the same edge-connected object.

The rest of the faces in the edge graph embedding are each tightly circumscribed by one edge-cycle that does not contain any other cycle from its same edge-connected object (they may contain separate edge-connected objects). These are called atomic edge cycles. The pumpkin in Figure 3.1 has eight atomic edge-cycles, including a tiny one at the tip of the stem, and one perimeter edge-cycle in purple.

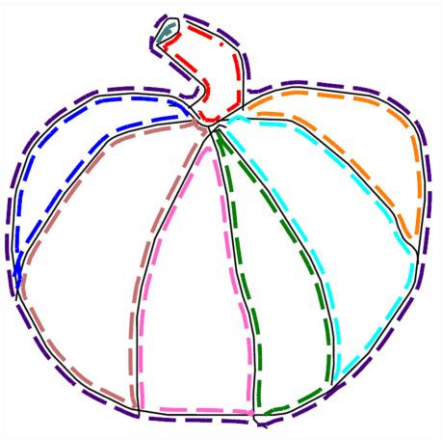


Figure 3.1: A pumpkin sketch with atomic and perimeter edge-cycles shown, traced from the edge graph embedding in Figure 2.7.

Perimeter and atomic edge cycles were originally introduced in (McLure et al. 2011). They are notable for several reasons. They trace closed paths around a jointly exhaustive and pair-wise disjoint set of regions that make up the sketch. The number of faces in the embedding is one more than the number of atomic edge-cycles. This allows us to validate the planarity of the entire ink-decomposition, using Euler's formula, which

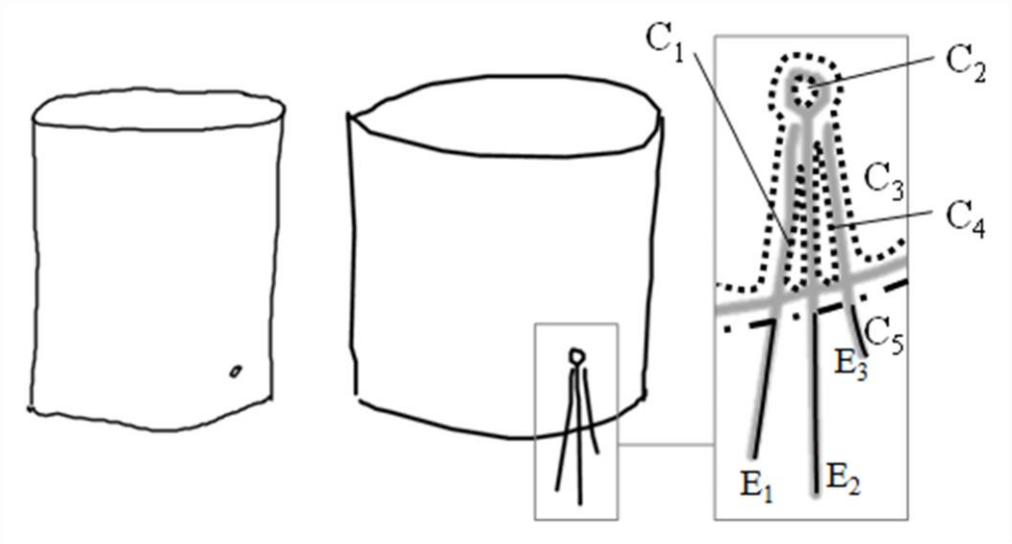


Figure 3.2: Two participants' sketches of cylinders from the Sun Up to Sun Down dataset, both drawn with water spouts.

defines the number of faces for planar graphs as a function of edges, vertices (junctions), and connected components (edge-connected objects). To fail signals an error resulting from aggressive resampling in the preprocessing described in Section 2.2.1, and triggers re-decomposition with higher fidelity. In line drawings of 3-dimensional opaque objects, atomic edge-cycles often circumscribe physical surfaces, and perimeter edge-cycles correspond to silhouettes. Note that the dual graph embedding for the edge graph embedding has nodes corresponding to the atomic edge-cycles and the background region, and edges connecting pairs of regions whose edge-cycles share an edge (with the dual edge crossing the shared edge). Therefore, this atomic edge-cycle embedding is also planar, and can serve as a substrate for efficient computations such as texture detection (McLure et al. 2015b; Section 6.2.2). Note that each *directed edge* – an edge traversed in one of its two directions, from a starting junction to an ending junction – appears in the trace for one and only one atomic or perimeter edge-cycle,

assuming the cycle's edge path traces around protrusions (for example, the three terminal edges that dangle off the bottom of the cylinder in Figure 3.2, or the body of a stick figure). This is leveraged when detecting cycles, but for characterizing shape, versions of the cycles are used with the protrusions pruned (as described later in this section), forming curvilinear polygons.

The algorithm for detecting these cycles, outlined in pseudocode in Figure 3.3, takes as input a set of edge choices (directed edges in which to search for cycles, initially all of the directed edges from the graph embedding), a set of valid starting edges (initially equivalent to the set of edge choices), a preference function (maximal turning preference vs good continuation preference), a Boolean flag that determines whether a cycle can traverse the same junction more than once, and a variable for accumulating

```
FindAtomicAndPerimeterCycles(edges E)
  ;; Finds atomic and perimeter edge-cycles by using a maximal
     turning preference and tracing around protrusions.
  FindCycles(E, E, RightmostTurningEdge, true, Ø)

FindSuperEdgeCycles(edges E)
  ;; Finds maximally turning cycles in the edge graph where
     super edges replace their sub edges.
  E_R ← ReplaceSubEdgesWithSuperEdges(E)
  FindCycles(E_R, E_R, RightmostTurningEdge, false, Ø)

FindContinuousCycles(edges E)
  ;; Finds cycles by following the most continuous edge at every
     junction
  FindCycles(E, E, MostContinuousEdge, false, Ø)

FindCycles(edge choices E, starting edges E_S, pref. function f_P,
           allow repeat junctions (Boolean) rjct, cycles C)
  ;; Returns a set of cycles found
  if E_S = Ø        ;; If there are no more starting edges,
     return C       ;;    then return the cycles found. (base case)
  e ← pop(E_S)      ;; Pick any starting edge.
  {c, E_V} ← TraceCyclePath(e, E, E_S, f_P, rjct, [e])
  C ← C ∪ {c}       ;; If a cycle was found, add it to the set.
  E_S ← E_S \ E_V   ;; Eliminate the visited edges as starting edges.
  FindCycles(E, E_S, f_P, rjct, C)   ;; tail-recurse

TraceCyclePath(edge e, edge choices E, starting edges E_S, pref.
               function f_P, allow repeat junctions (Boolean)
               rjct, path P)
  ;; Returns a cycle if one is found, along with all visited edges.
  e_next ← call(f_P, {e, E})  ;; select the next edge using the pref. fn.
  ;; If the path already contains the next edge, then a cycle has
     been found. Return it along with all visited edges.
  if (e_next ∈ P)
     return {SubsequenceStartingWith(P, e_next), P}
  ;; If there is no next edge, or repeated junctions are prohibited
     and we encounter one, or if the next edge is not among the
     valid starting edges, then fail
  if ( ¬e_next ∨
     (¬rjct ^ (Junctions(e_next) ∩ Junctions(P))  or
     ¬(e_next ∈ E_S) )
     return {null, null}
  ;; Otherwise, add the next edge to the path and tail-recurse.
  P ← P + e_next
  TraceCyclePath(e_next, E, f_P, rjct, P)
```

Figure 3.3: Procedures for tracing edge-cycles. All edges are assumed to be directed edges (two per edge in the edge graph).

cycles (initially an empty set). A starting edge is popped off the set of valid starting edges. This directed edge, along with the set of edge choices, the valid starting edges, the preference function, the repeat-junction flag, and a sequence of visited edges called the path (initially containing the initial edge) are passed to the path-tracing function. This function is expected to return a new cycle path if one is found, and a set of the directed edges that were visited in the search. If a cycle is found, the visited edges are eliminated as starting edges since they will only lead to discovering the same cycle. The cycle-finding function is then called in a tail recursive fashion. When searching for atomic and perimeter cycles, the path-tracing procedure is guaranteed to find a cycle without visiting any extra edges not in the cycle, which means the overall cycle-finding procedure will check every edge exactly once.

The path-tracing procedure first selects a next edge by applying the preference function to the current edge and the set of edge choices. For perimeter and atomic edge-cycles, a maximal turning preference is used, so the next directed edge is the one associated with the rightmost-turning boundary point that is also among the choices, measured from the boundary point associated with the current edge at its ending junction. The rightmost turn is the one with the largest clockwise angular distance between the junction-referenced orientation of the current directed edge's ending boundary point and the junction-referenced orientation of the next directed edge's starting boundary point. (Note that we do not determine the rightmost turn using the local edge orientations of the boundary points. In Figure 3.4, the maximal right turn when coming in along the green edge is the purple edge, even though the maximally right-turning local

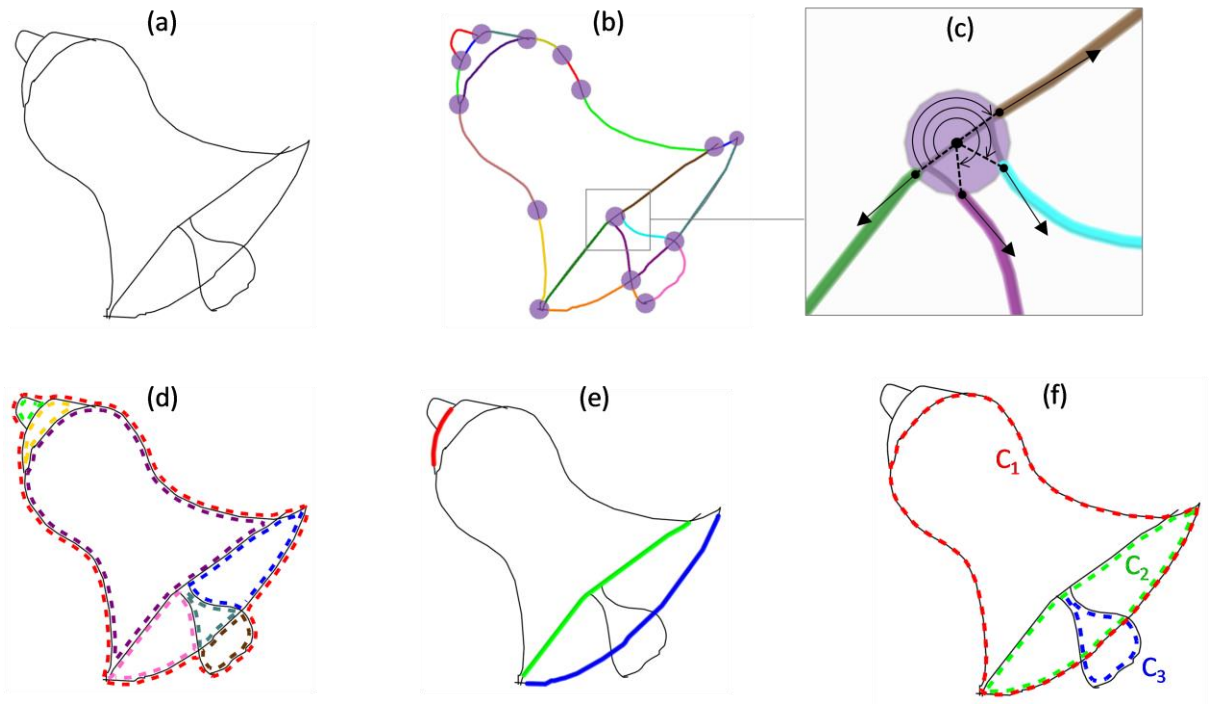


Figure 3.4: (a) A sketch of a bell from (Eitz et al. 2012). (b) The edges and junctions found during ink decomposition. (c) The four boundary points along a junction in the bell (d) Its atomic and perimeter edge cycles. (e) Three detected super-edges. (f) Its continuity-based edge-cycles, with C_1 generated by tracing with the continuity preference, and C_2 and C_3 generated by tracing using super-edges and a maximal turning preference.

edge orientation is the cyan edge.) The (end) boundary point of the current edge is an option, so if there are no other edges connected to the junction besides the current edge – as is the case at a terminal edge like the red edge in the rabbit's leg in Figure 2.4 or the dangling water trails hanging off the bottom of the cylinder in Figure 3.2 (edges E_1, E_2, and E_3) – then the next directed edge will be the reversed version of the current directed edge. If the next edge is an edge in the current path, a cycle has been traced from this edge back to itself, through everything after it in the current path. The subsequence of edges that makes up the cycle is returned as the found cycle and all of the current path is returned as the edges visited. If the next edge is not a valid starting edge, or if a repeated junction is found while they are prohibited, path-tracing fails

to find a cycle. It returns null for both the found cycle and the visited edges (a cycle may still be found using edges in the path except for the first, since for other preference functions, there may be two edges among the choices that have the same next edge). Otherwise, the path-tracing function is called (tail) recursively on the next edge, adding the next-edge to the path.

When repeated junctions are allowed and a maximal turning preference is used, as when detecting perimeter and atomic cycles, the cycle paths will trace along protrusions, following them out away from the cycle and then back in the other direction. Connected subgraphs that are protrusions are always trees. They are detectable as a subsequence of directed edges in the cycle whose corresponding undirected edges, together with its node subsequence, forms a pre-order tree traversal (with duplicates). These protrusions are subtracted from the edge-cycle when computing qualitative shape representations in order to focus on the curvilinear polygon component of the edge-cycle, but their presence is noted with edge-cycle attributes `EdgeProtrudedShape`/`EdgeIntrudedShape` for perimeter and atomic cycles, respectively.

3.1.2 Continuity cycles

Edge-cycles are found in two additional way that incorporate good continuation. Both methods use the same recursive path tracing function above. To ensure that the continuity cycles found are simple curvilinear polygons (no intersections), no repeated junctions are allowed.

The first method simply applies the maximal turning preferences to a set of edge choices in which any detected super edges replace their member edges. The super edge's start junction is the start junction of the first edge in the sequence of member edges, and its end junction is the end junction of the last member. Any new edge-cycles found are continuity cycles. For

example, in Figure 3.4(f), C_2 traces along two super edges (plus one regular edge) using a maximal turning preference. Note that continuity cycles found in this way need not be edge-cycles that contain super edges. Some may simply result from removing the sub-edges, like cycle C_3 in Figure 3.4(f), which results from replacing the member edges of the blue super edge with the blue super edge, thereby eliminating the members as turning options in tracing C_3.

Finally, we add in cycles that trace the edge paths with a preference for good continuation. We use the same procedure as above (without replacing edges with super edges) and pass in a heuristic continuity preference function. Specifically, the function used here, given an (incoming) edge, finds the outgoing edge leaving its end junction with lowest *continuity difference*, a sum of four terms:

$$\Delta Cont = \Delta Orientation + \Delta Curve + Jog + SeparateStrokePenalty$$

We refer to the incoming edge's ending boundary point as b_1, and the outgoing edge's starting boundary point as b_2. Θ_{in} is the incoming orientation – the inverse of the local edge orientation of b_1. Θ_{out} is the outgoing orientation – the local edge orientation of b_2. $\Delta Orientation$ is the difference in orientation between Θ_{in} and Θ_{out}. $\Delta Curve$ measures the absolute difference between the cube-root of the incoming edge's average curvature C_1 and the cube-root of the outgoing edge's average curvature C_2. (Curvature, the inverse of radius, is signed to reflect the direction of curvature and has no real bounds. The cube-root preserves the sign while scaling down the difference). Jog measures the "jog" between the boundary points – informally, correlated to how much the hypothetical contour connecting the incoming edge to the outgoing edge is laterally displaced while passing through the junction. We call the orientation of the vector between b_1 and b_2 the jog orientation Θ_{jog}. Jog is the average of the angular

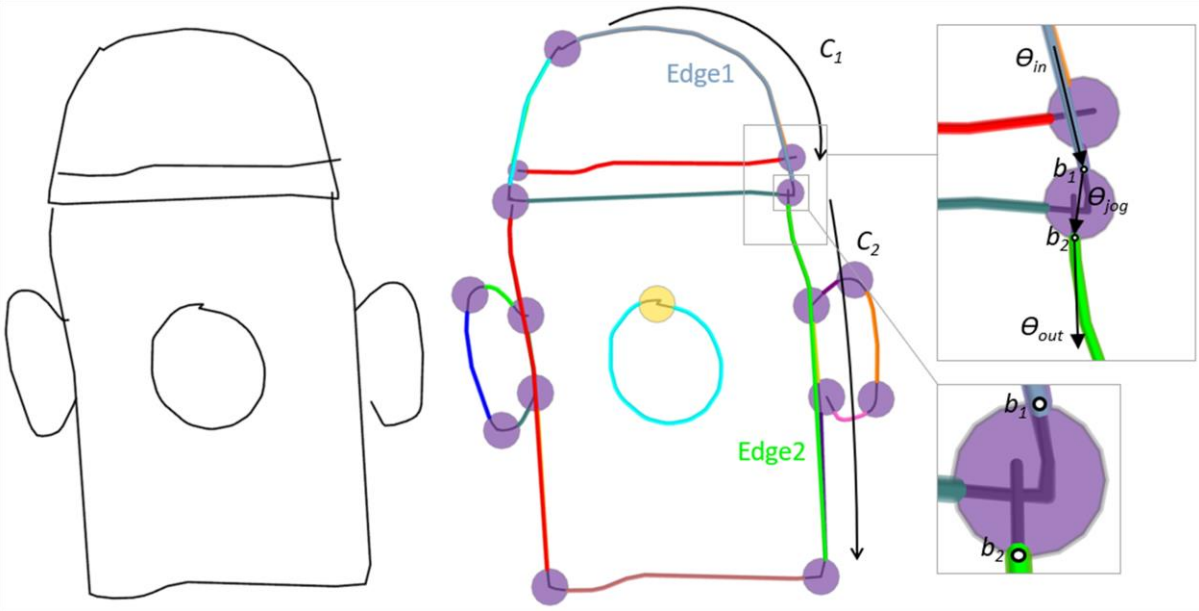


Figure 3.5: A sketch of a fire hydrant from (Eitz et al. 2012), and to its right, its decomposition into edges and junctions. The super edges, including Edge1 and Edge2, are overlaid (over their atomic edges). On the far right are close-up views of the junction between Edge1 and Edge2.

displacement between Θ_{in} and Θ_{jog} and the angular displacement between Θ_{jog} and Θ_{out}. $SeparateStrokePenalty$ is meant to give a boost to edges that came from the same stroke or locally overlapping strokes in the original ink. The penalty is 0 if b_1 and b_2 lie on the same stroke in the original ink. It is $\frac{\pi}{2}$ if the original ink stokes corresponding to b_1 and b_2 intersect inside the junction. It is π otherwise. All of these necessary elements for computing the continuity difference are shown in Figure 3.5 for assessing the continuity between Edge1 and Edge2. In this case b_1 and b_2 do not come from the same original ink stroke but their original ink strokes do intersect inside the junction (bottom right in the figure).

3.1.3 Vocabulary Extensions

Perimeter, atomic, and continuity edge cycles, as curvilinear polygons, are reified as entities and described using the shape-level vocabulary in CogSketch (Section 2.2). This section describes the additional relations and attributes that are specific to edge-cycles.

Two attributes introduced at the end of Section 3.1.1, `EdgeProtrudedShape` and `EdgeIntrudedShape`, mark edge-cycles that have protrusions on their outside (for perimeters) or inside, respectively. There are also attributes to mark the type of edge-cycle – `PerimeterEdgeCycle`, `AtomicEdgeCycle`, or `ContinuityEdgeCycle`. `boundingEdgeFor` is a relation that ties each edge-cycle to each of its member edges. Edge-connected objects are marked with the `EdgeConnectedObject` attribute.

Intersection relationships between edge-cycles are important because they signal connectivity in what they depict. Edge-cycles that share an edge are related by `sharesEdgesWith`, and any two edge cycles that share either a junction or an edge are related by `touchesDirectly`, which provides a notion of adjacency. Positional relations and relative-orientation relations are only computed by default between adjacent edge-cycles. A third intersection relation, `edgeSubsetOf`, is used in the case when all of the edges in one cycle form a subset of those in another cycle.

As an example, the close-up in Figure 3.2 (right) in shows the atomic (C1-C4) and perimeter (C5) edge-cycles in the vicinity of the stream depiction. The symmetric `sharesEdgesWith` relation holds between the pairs: {C1: C4}, {C1: C3}, {C1: C5}, {C4: C3}, {C4: C5}, {C3: C5} {C3: C2}. The `touchesDirectly` relation, also symmetric, holds between all of these same pairs as well as {C1: C2} and {C4: C2}. C2 is related to C3 by `edgeSubsetOf`, since all of

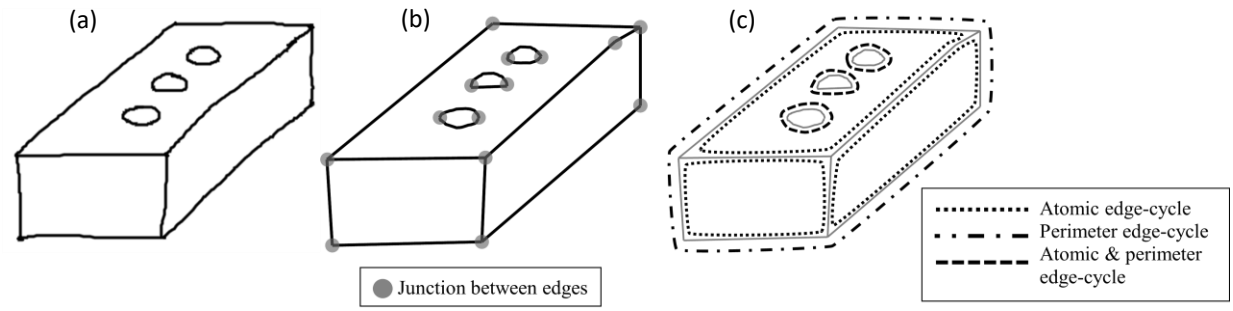


Figure 3.6: (a) A sketch of a brick from the Sun Up to Sun Down dataset, (b) its junctions, and (c) its edge-cycles.

the edges that comprise C2 (in this case only one edge) are also in C3. The three bottom edges E1, E2, and E3 – protrusions found when tracing the perimeter cycle C5 – are not part of any edge-cycle (taken as a curvilinear polygon) because they terminate at one end, but they add an `edgeProtrudedShape` attribute to the perimeter edge-cycle C5.

Containment relationships between atomic edge-cycles and edge-connected objects are important for encoding nested imagery within a sketch. The `atomicPartOf` relation connects an atomic cycle to the edge-connected object it is part of. Conversely, the `containsObject` relation holds between an atomic edge-cycle and an edge-connected object if the cycle spatially contains the object without sharing any edges or junctions with it (e.g. the brick's holes in Figure 3.6).

The between relation is queried for some triples of edge-cycles. Specifically, we query whether B is between A and C if A shares a junction with B and B shares a junction with C, they are all different cycles, and none of them overlap. Our notion of betweenness for 2-D closed shapes borrows from the ideas in (Bloch, Colliot, and Cesar, 2006), which makes use of the convex hull. In our version, B is between A and C if it lies completely inside their convex hull or bisects their convex hull, with the exception of situations where the perimeter of B is mostly

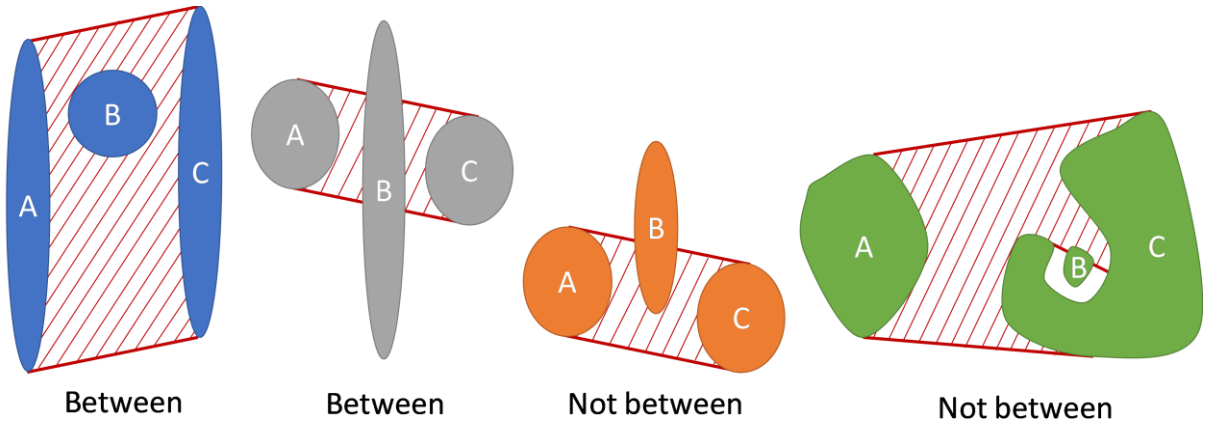


Figure 3.7: Positive and negative examples of the betweenness relation between triples of 2-D closed shapes.

(>50%) occluded by one shape from every point along the perimeter of the other shape. Resampling is used on the perimeters to keep computation tractable. Figure 3.7 shows positive and negative examples of our version betweenness. From left to right, (blue) B is between A and C because B lies completely inside the convex hull of A and C, (gray) B is between A and C because B bisects the convex hull of A and C, (orange) B is not between A and C because it does not bisect the convex hull or lie completely within it, and (green) B is not between A and C because, despite lying inside their convex hull, most of the perimeter of B is occluded by C from every point along the perimeter of A. On fuzzy decisions like the third (orange) case in Figure 3.7, our criteria are designed to err on the negative side, since we would like to keep cases concise. Bloch et al. (2006), on the other hand, treat betweenness as a fuzzy relationship by computing a quantitative degree of betweenness. They also have more sophisticated handling for cases when A is much more elongated than C or vice versa.

Betweenness is an exotic positional relationship, in that it is ternary, symmetric in two of its arguments, and invariant to rotation and reflection. The between relation was not used in

Experiment 1 below because it was added later. It was used for all subsequent experiments that used edge-cycle representations (Experiments 2 and 4).

3.2 Mixing edges and edge-cycles

Edges are an effective basis for perceptual organization for many visually simple sketched objects. As sketched objects get more complex, the number of edges can become overwhelming. Grouping edges into edge-cycles can result in a more manageable number of more abstract visual elements. Closed shapes are a useful type of visual element, especially in depictions of physical objects that include surfaces, holes, and concavities. In many sketched objects, the contours of these closed shapes, expressed in a sequence of constant curvature edges, are not as reliably similar across examples from a category as the relative position, topology, and coarser grained properties of the shapes. For example, the larger regions of whitespace in the house and barn in Figure 3.8 (silhouette/walls/roof) are similar in terms of their eccentricity, orientation, size, relative position, and convexity. The organization and visual properties of the edges in these shapes are not as similar, with changes in the number of edges, curvature, and discontinuities. Attending to closed shapes has the potential to reduce the complexity of analogical mappings while producing

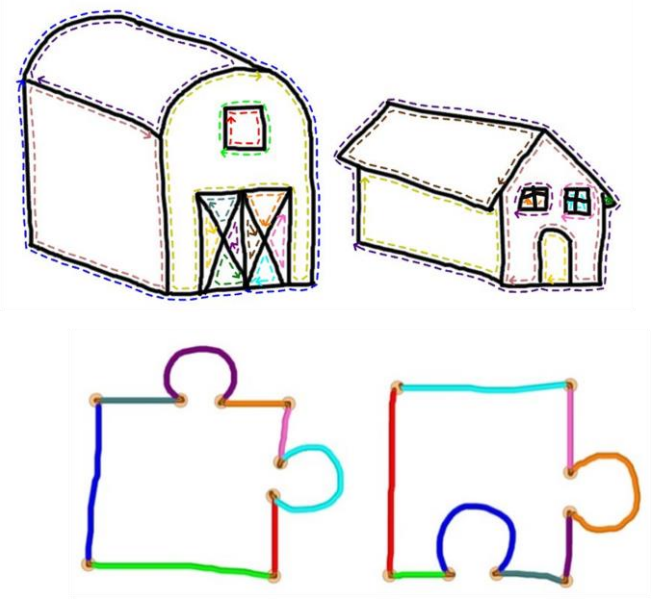


Figure 3.8: Simple shapes may be best captured at the edge level, whereas edge-cycles may be more stable and tractable for more intricate sketched objects. Here, colors depict separate entities, not an analogical mapping.

more stable representations for classification. Of course, exclusively looking at edge-cycles will miss opportunities to discriminate patterns that hinge on the shape of individual edges (e.g.

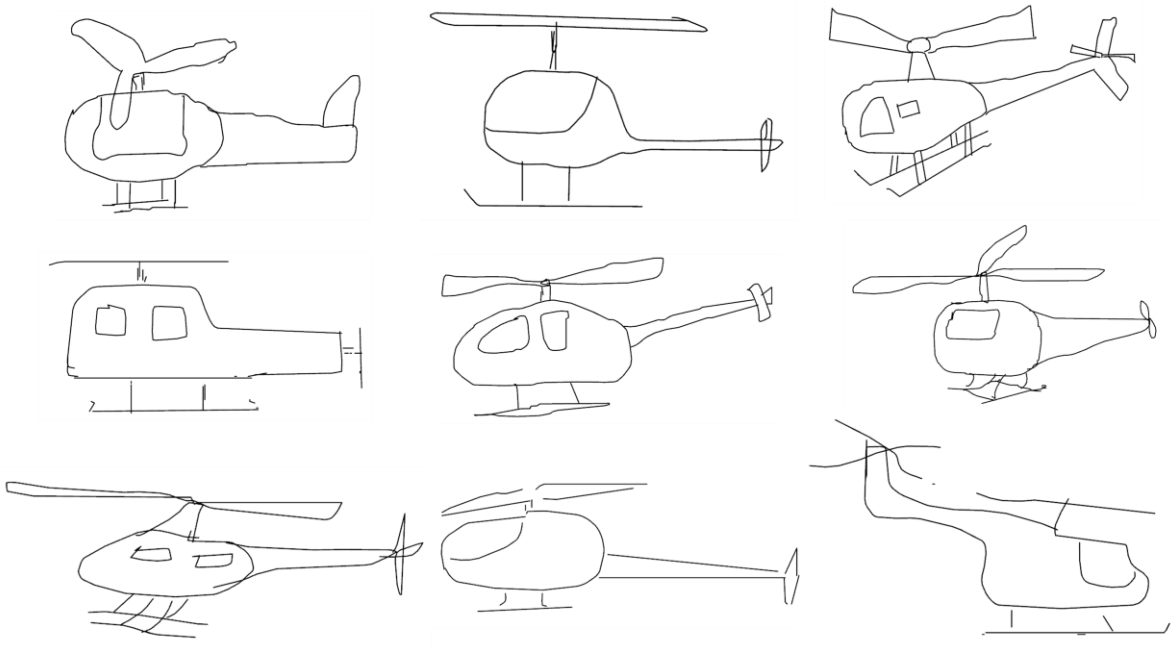


Figure 3.9: Sketches of helicopters from (Eitz et al. 2012).

discriminating between the puzzle pieces in Figure 3.8), or on any aspects of the open parts of the shape, which cannot be described as composites of closed shapes (e.g. the bodies of stick figures).

Edges and edge-cycles may be complimentary in describing different parts of the same sketch. For example, helicopters (Figure 3.9) tend to be drawn with chassis (cockpit/tail) that are relatively consistent at the edge-cycle level while taking on differently shaped contours. On the other hand, the feet of the helicopter tend to consist of one or two long horizontal edges representing the rail-like feet, connected to the chassis via differently imagined structural configurations. These alternative configurations, in concert with alternative views and thin-line objects (whose faces in the edge graph do not correspond to opaque surfaces, but negative

space), result in edge-cycle arrangements that are less stable than the presence, orientation, position and length of the edges representing the feet.

Relation in (<relation> <cycle> <edge>)	**Holds when…**
boundingEdgeFor	<edge> is an edge (or super edge) in the <cycle> path.
edgeOverlapsObjectInk	<edge> is not fully in the path, but it has at least one sub-edge in the <cycle> path.
elementsIntersect	At least one of the junctions connecting to <edge> or its sub-edges lie on the <cycle> path, without sharing any of its sub-edges.
edgeContactsFromInside	<edge> overlaps or intersects <cycle>, and at least one of its sub-edges (or itself) lies in interior region of <cycle>.
edgeContactsFromOutside	<edge> overlaps or intersects <cycle>, and at least one of its sub-edges (or itself) lies in the exterior region of <cycle>.

Table 3: The five relations that relate edge-cycles to the edges they contact.

In addition to edges and edge-cycles separately, we examine an encoding scheme that combines the edge and edge-cycle assertions for a sketched object. Including edge and edge-cycle descriptions to cases should allow SME to match edge patterns where edge patterns are consistent and cycle patterns where cycles are consistent. Representational connective tissue is added to relate edge-cycles to edges. The complete set of these cycle→edge relations are shown in Table 3. This connective tissue allows edge-level correspondences to influence edge-cycle-level correspondences and vice versa. It also facilitates entity-based filtering, described below.

The satellite dishes in Figure 3.10 demonstrate the advantage of allowing edge and edge-cycle correspondences influence one another. The center and right images are alternative renderings of the same example, with different subsets of its perceptual entities shown. When using an encoding scheme based purely on edge-cycles, SME maps the four edge-cycles on the left (blue, red, green and pink) to the edge-cycles with the corresponding colors in the center

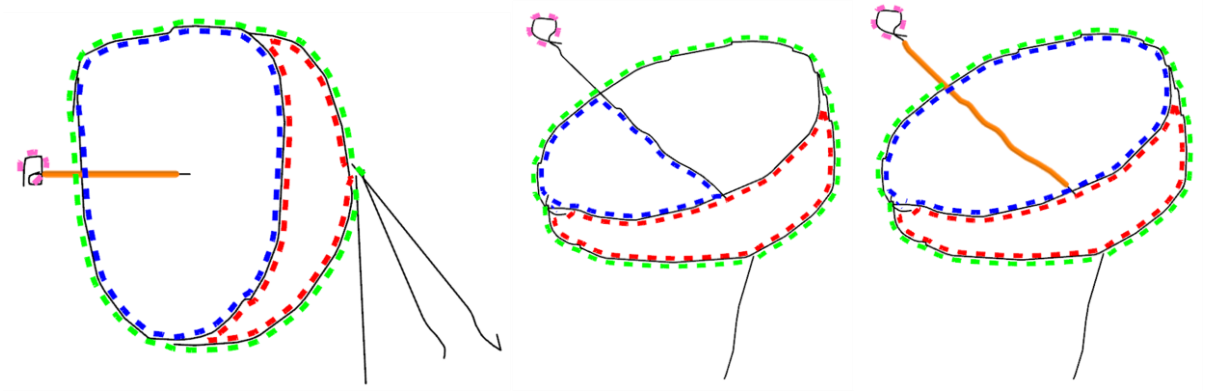


Figure 3.10: Two examples of a satellite dish from (Eitz et al. 2012).

image. The blue cycles in the left and center images, while not intuitively analogous, have many similar shape attributes – e.g. elliptical variance, compactness – due to the significant curvature discontinuity in the top left region of the dish on the left. When the hybrid edge/edge-cycle encoding scheme is used, the orange edge on the left is mapped to the orange edge on the right. Because these analogous edges are connected to the edge-cycles via various contact relationships, SME is able to find a larger shared system of relationships (a better analogy) by instead mapping the blue cycle on the left to the blue cycle on the right, thereby overcoming the misleading shape similarities. Note that because of the systematicity bias in SME, which prefers to match larger systems of relationships, the highly alignable pink edge-cycles are indirectly contributing to this correction by strengthening the correspondence between the orange edges (again via their cycle→edge topological relationships, this time with the corresponding pink cycles).

This hybrid encoding scheme includes an analogical constraint to exclude cross-partition correspondences between edges and cycles, (meaning the mapping can't have edges mapping to edge-cycles). Keeping case size constant, this constrains the matching, speeding it up – because fewer match hypotheses form – and potentially helping it resist getting derailed by too much

detail. The down side is missing opportunities to match an edge to an edge-cycle where appropriate. Some of the helicopters in Figure 3.9 (e.g. center and center-left) have rotors that might align nicely if cross-partition correspondences were allowed.

Note that continuity cycles, perimeter cycles, atomic cycles, edges and super edges, and different flavors of sub-part relationships connecting them provide an explicit representation of hierarchical visual structure. We take the view that there is no intrinsic issue with including overlapping or subsumed visual entities in the same qualitative representation – in fact, SAGE over such representations should be able to learn statistics about how often different parts of the hierarchy manifest.

On the other hand, the key to success is not simply throwing more and more into the input cases for SME – a tractable, non-optimal graph matcher – expecting it to find better and better matches. This is to say nothing of the match being too time intensive to be of any real use. Figure 3.11 shows two similar, reasonably simple examples of the concept *Table,* as well as a

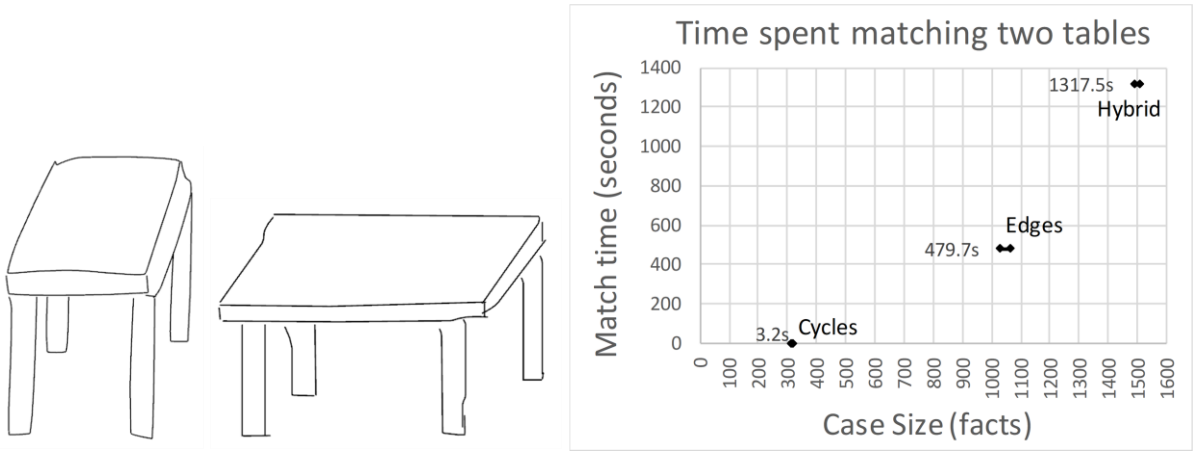


Figure 3.11: (Left) Two sketches of tables from (Eitz et al. 2012). (Right) Six cases – two tables crossed with three encoding schemes – are plotted on the x-axis by case size and grouped by lines into three pairs. Each pair represents a match. These pairs are plotted on the y-axis by how much real time the match took.

chart of the time taken to perform a match between them using the full representations generated by the edge-cycle, edge, and hybrid schemes. These representations contain ~300, ~1000, and ~1500 facts respectively (for each of the two tables), and take ~3 seconds, ~8 minutes, and ~22 minutes, respectively, on an Intel i7-4790, 3.6GHz CPU with 16 GB of RAM. The next section describes strategies for filtering these representations to be more tractable, and Experiment 2 below examines the effectiveness of these three encoding schemes under filtering conditions that constrain the case size.

3.3 Detail as a Trade-off: Discriminability versus Tractability

The Structure Mapping Engine (SME; Section 2.1.2) puts elements into correspondence by matching entire systems of relationships at a time to optimize systematicity. This matching step amounts to the maximum common induced subgraph problem for labeled graphs, which is NP-hard, and even when it's performed greedily to avoid non-polynomial computation as in SME ($O(n^2 \log(n))$), can be prohibitively expensive when the inputs are large enough to demand resources (time or computation) that exceed what is reasonably available in the task context. One the other hand, larger inputs potentially include more useful information for discriminating concepts, either by including more characteristic or critical properties of concepts, or more subtly, by providing important relationships for finding more productive mappings. A fundamental trade-off in a real-world system learning via graph matching between structured inputs is between including enough information to generate useful analogies and limiting input size to stay tractable.

Some broad questions the experiments in this chapter address are, what are possible strategies for putting hard limits on the size of visually encoded input to generate better

mappings under resource constraints? How do these strategies perform in a learning task, and how does this performance interact with the number of training examples? What is the effect of increasing input size on performance?

By looking at the effects of different case-size limits across multiple filtering strategies, we are also gaining insight into other questions about analogical learning. Framing case size as a simple trade-off between discriminability and tractability suggests that when resources are not an issue, larger inputs will tend to perform better. Suppose increasing case size does not monotonically improve classification performance. This would provide evidence for the idea that the analogical match can be led astray by inputs that are too large, which is consistent with Finlayson and Winston's goldilocks hypothesis (2006). Will the learning task performance curves for varying case size have peaks, suggesting that for a given combination of encoding scheme and filtering strategy, there is some optimal case size, beyond which larger inputs stop being useful even when resources are a non-issue? Will this optimum look stable across different filtering strategies? Across different encoding schemes? Any of these outcomes would tell us something more general about optimal case sizing for analogical learning using SME over perceptual input. Even without them, the experiments in this section provide evidence about how case size interacts with filtering strategies and encoding schemes.

3.3.1 Fixed Size Representation Filters

One way to put an upper bound on computing resources when using SME is to put an upper bound on match hypotheses, and one way to do that is to put an upper bound on the number of facts in any case that will be input to SME.

Enforcing a hard limit on the number of facts in every case by simply removing some of them can cause errant hypotheses to emerge during analogical learning because a comparison of two cases encoded with the same scheme may produce misleading differences. We call this a *filter error*. A candidate inference about an entity in the target may have been determined not to hold while encoding the target (a true difference), or it may have been filtered from the target while the otherwise corresponding assertion about the corresponding base entity had not been filtered from the base (a false difference). Of course, there is no way to know prior to the match that these base and target entities would be placed in correspondence, which makes it difficult to filter facts from both sides in such a way that guarantees filter errors won't occur.

One way to make such a guarantee is to filter all facts of a given predicate from both base and target at once. This works because of the Structure Mapping constraint of tiered identicality, which requires that for two expressions to correspond, they must share the same predicate. The problem is that if filtering depends on the match, then the unfiltered cases must be stored in long term memory. Either that, or one must use the same filter on the entire set of cases that ever might be compared. Because the number of entities is unpredictable, this would put no guarantees on case size. Trying to accommodate the largest cases (those with the most entities) would likely result in most cases being over-filtered, lacking relevant details.

Maintaining two layers of representation – one economical and one non-economical – may provide opportunities for analogical learning that are more resistant to filter errors and whose computational tractability can be controlled. The analogical matching process could be given an economical representation of any size, as this is the computationally expensive step. The non-economical representation could be used to validate the differences found by the filtered match.

Of course, this still requires storing the uneconomical representations in long term memory, which is its own tractability issue. We leave this approach for future work.

In this chapter we experiment with approaches that filter asymmetrically, and only keep around the economical representation, and therefore are vulnerable to filter errors. Each case is filtered based on properties of the case alone, independent of other cases. Like Lovett et al. (2007), we consider two types of potential signals for filtering – entity preferences and predicate preferences. Our specific preferences differ from theirs, partially because they only focused on the edge-level, but there are some common intuitions behind the entity-based filters. That work is further discussed in Section 6.2.1.

Here, entity-based filters construct a case of a limited size by incrementally selecting the next best entity (determined by the utility measure described in the next section) and pulling into the case all facts that mention the entity without mentioning entities that have not been selected yet. This greedy step repeats until the case limit is reached, at which point the facts pertaining to the last selected entity are truncated to fill up the case to its limit.

Predicate-based filters work the same way, except they pull in facts by incrementally selecting predicates instead of entities, using a simple utility measure provided in Section 3.3.3 below.

3.3.2 Visual coverage: A bottom-up heuristic for entity-based filtering

Knowing nothing about what other cases a case will be compared to, we are interested in using qualitative representations that capture the visual information in the quantitative input as efficiently as possible. Visually efficient representations represent more of the visual input with fewer facts.

Our entity-based filters for visual efficiency select the entity that describes as much of the visual input as possible, while prioritizing visual input that is underrepresented by the rest of the case. The motivation for attending to larger objects is two-fold: (a) Large objects characterize more of the visual input at once, and (b) false positive entities resulting from noise and unimportant decorations – entities that would be helpful to ignore – tend to be small. The motivation for attending to the underrepresented is to take a breadth-first approach to capturing the visual field. Therefore, we treat large objects with low overlap as the place to start in building out a qualitative representation, and as space allows, we drill down into more and more of their detail.

Importantly, all of the information needed to select an entity using these filtering strategies is available in the case itself (sans ink), because we estimate visual input coverage using qualitative, relative size predicates that are standard at the edge and shape levels of representation in CogSketch, and because our methods for organizing edges and shapes leave us enough information about composition to estimate overlap from the qualitative representation alone. This allows us to apply filters to cases in long term memory after the ink they encoded is no longer available. Another important benefit not tested in this work is that the filter can potentially be applied to cases for which ink has never existed, such as a SAGE generalization or a case that is otherwise imagined, perhaps by combining concepts.

Our two entity filters look at two different ways of framing visual coverage for sketched objects. The first focusses on the ink. The second focuses on the area of the sketching plane.

The first strategy is the *ink coverage filter*, which estimates the total length of the ink polylines that an entity covers. The ink coverage of an edge is captured approximately and

directly by its relative length attributes (`lengthTiny`, `lengthShort`, `lengthMedium`, `lengthLong`). The ink coverage of an edge-cycle can be estimated by adding up the approximate lengths of its atomic (non-super) bounding edges. A super-edge is linked to its sub-edges in the edge-level representation and can similarly have its length estimated by adding up the approximate lengths of the sub-edges. The same measure can be used on glyphs or edge-connected-objects, given a chain of relationships that link them to their edges and cycles. The length of an edge-cycle or a super edge could be measured and used directly, but by going through atomic edges instead, we can estimate the amount of ink shared between entities of any of these types. Note that the atomic edges and junctions in the edge graph embedding cover the ink in the sketch in a fashion that is jointly exhaustive and pair-wise disjoint (JEPD). We can target underrepresented parts of the sketch by targeting underrepresented atomic edges.

The *utility* of an entity (used by the greedy entity selection process from the last section) is treated as the summed utilities of its atomic edges. An edge's utility is equal to the edge's size (quantified), discounted (divided) by how many selected entities would be covering the atomic edge if this entity were selected, i.e. the number of heretofore selected entities covering the atomic edge, plus one.

$$Utility(e) = \sum_{a \,\epsilon\, Atomic(e)} \frac{Size(a)}{|SelectedCoveringEntities(a)| + 1}$$

Thus, an atomic edge that has already been covered by one selected entity needs to be twice as long as a novel edge to be considered just as useful. An atomic edge covered by two selected entities needs to be thrice as long.

Figure 3.12 shows how qualitative sizes are quantified. CogSketch encodes entities' sizes by dividing the largest entity's size into 4 quartiles, but when assigning those quartiles to discrete sizes, it keeps clusters of similar sized entities together (e.g. yellow dots). The quantities assigned to each qualitative length are meant to line up with the centers of the quartiles. The overall scale doesn't matter for scoring ink coverage.

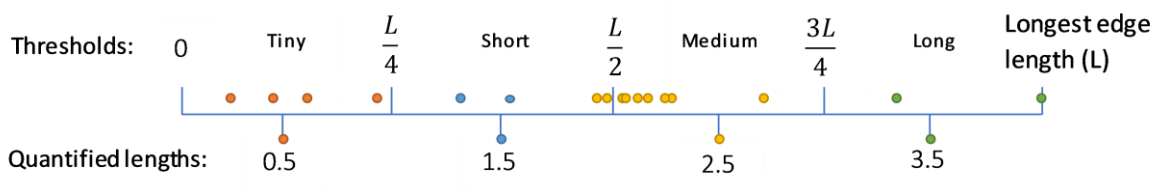


Figure 3.12: Encoding and decoding CogSketch's qualitative length attributes for edges. (The same schema applies to other attributes that measure qualitative degrees.)

The *area coverage filter* applies the same formula to the selection of closed shapes by their areas. Here, the atomic edge-cycles are the atomic elements used to determine size and overlap. Atomic edge-cycles, analogous to atomic edges, are nearly JEPD in the way they tile over enclosed regions in the sketch. Containment relationships were used to decompose cycles into their atomic cycle elements.

One caveat is that an atomic edge-cycle can contain an edge-connected-object (and therefore its cycles). The JEPD regions of whitespace in a sketch can have holes, but these regions are not accurately accounted for by the cycle vocabulary, which applies to curvilinear polygons. A more appropriate notion of region-area would fit better the spirit of the filter.

The area coverage was not implemented for the hybrid encoding scheme, because it wasn't clear how to characterize, from the knowledge level, the area that an edge covers in terms of whole atomic cycles. However, to that end, the hybrid representations used here would allow

one to find which edge-cycles a given edge participates in, estimate their areas, and estimate what fraction of each cycle's contour the edge represents.

3.3.2.1 A problem with filtering entities: What goes with nothing?

A drawback of purely entity-based filtering, given the qualitative representations used here (which include some very common attributes/relationships), and SME as an analogical matcher, is that many entities can end up completely blocked by the filter, and SME is willing to map as much as possible of what makes it through.

Consider the two instances of the category *Flower with Stem* from (Eitz et al. 2012) shown in Figure 3.13 (top), encoded using the hybrid edge/edge-cycle perceptual organization and the ink-coverage entity filter with a limit of 100 facts. Only the entities that made it through the filter are overlaid (one edge and four cycles on the left, and one edge and five cycles on the right). Colors reflect the correspondences found by SME between entities on the left and right. The black edge-cycle on the right is unmapped. Different relative sizes of the pedals and the center part of the flower lead to different entities being chosen by the filter. The elements that make it through, and whose best (unfiltered) analogs did not make it through, should go with nothing. They are nonetheless paired up into the best

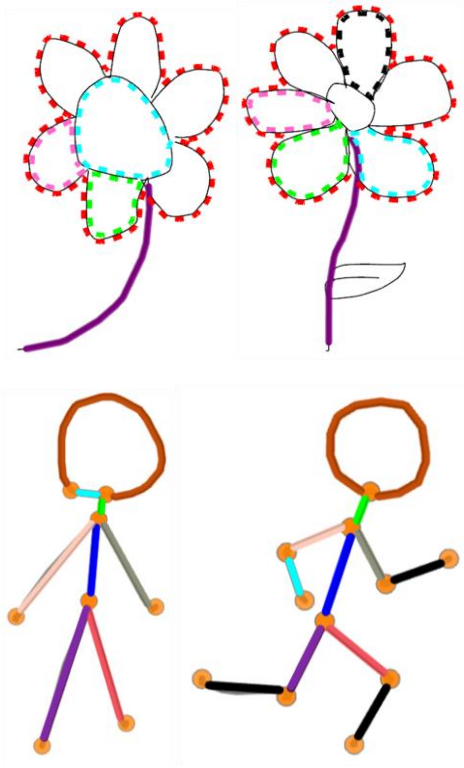


Figure 3.13: Two instances of the *what goes with nothing* problem.

correspondences possible. For example, the cycles shown in light blue (center part in the left flower, lower right pedal in the right flower) are put into correspondence for minor commonalities in their shape attributes and relationships to nearby entities, but each of the two would have had a different, much stronger analog in an unfiltered match.

The stick figures in the bottom row of Figure 3.13 show an instance of the *what goes with nothing* problem that does not result from filtering. Again, colors mark edges found by SME to correspond and black edges are unmapped. The extra (light blue) edge resulting from an unintended junction in the head on the left has no good analog on the right, but given some representational overlap (e.g. it is a straight edge), SME readily maps it to one of the extra (intended) edges on the right. This scenario exhibits the same basic dilemma as before: extra entities in base and target resulting from independent phenomena, which get put into correspondence because it marginally strengthens the match.

To see why this issue is so prevalent with entity-based filtering, imagine base and target cases that each have four entities in their unfiltered forms, and imagine that two entities make it into each (independently) filtered case. It will only take *some* overlap in their attributes and relations for SME to prefer to match both entities in the filtered base to both entities in the filtered target (in whichever pairing), over leaving any entities unmapped. This overlap may come from an attribute or relation that is ubiquitous enough to be common to all 8 entities in the cases (e.g. (isa <x> AtomicEdgeCycle)) or it may come from noise. It may even come from a dimension of legitimate, salient similarity, but not one that would win out as a correspondence in the unfiltered comparison. The problem is, given no other information, the chances of choosing four entities – two from each case – that happen to form two correspondences in the filtered

match that would also be in the unfiltered match is one in six ($\binom{m}{n}$ for *n* entities in the filtered case and *m* entities in the unfiltered case[3]). So there is a five-in-six chance that SME will be left to choose what goes with what among entities for which some should go with nothing – either because their strongly analogous entity was filtered, like in Figure 3.13(top), or because they have no analog even in the unfiltered cases, like in Figure 3.13 (bottom).

An appropriate entity filter could improve these odds, as would widening the filter of course. But it may suggest that the entities should not be filtered, at least not directly. In the next section we examine a filter that instead filters using predicates. This should tend to filter far fewer entities, while incorporating less information about each selected entity.

3.3.3 Predicate rarity: A bottom-up heuristic for predicate-based filtering

Again, imagine our situation from the last section in which two cases containing four entities each are filtered and then compared using SME. Let's go further and assume that each filtered case contains *f* facts, and each unfiltered case contains *m* entities, and whose facts are all binary relations that are not reflexive, and over which the (possible pairs of) entities are evenly distributed. The chance a randomly chosen assertion *does not* take a given entity in either argument slot is $\left(\frac{m-1}{m}\right) \times \left(\frac{m-2}{m-1}\right) = \left(\frac{m-2}{m}\right)$. Repeatedly selecting *f* facts, *with replacement*, the chances that none of them mention a given entity is $\left(\frac{m-2}{m}\right)^f$. Assuming this occurring with one entity is mutually exclusive with it happening with another (this is not actually the case) the probability of it happening with *some* entity is $m\left(\frac{m-2}{m}\right)^f$. This is an upper bound on the actual

[3] There are $\binom{m}{n}^2$ ways to choose *n* entities from each set of *m*. There are $\binom{m}{n}$ ways to choose *n* correspondences out of the set of *m* correct correspondences.

probability, since filtering does not select facts with replacement (decreasing the chances of choosing *f* facts not containing a given entity), and since one entity being filtered completely is not mutually exclusive from some other entity being filtered completely (meaning the probability of at least one occurring is less than the sum of their probabilities). Therefore, for a sense of the order of magnitude, we can say that when choosing 100 facts from any sized unfiltered case, containing non-reflexive binary relations, across which four entities are mentioned in a way that is evenly distributed, the chances of an entity not making it through the filter is *less than* one in 3×10^{30}. Even with a fact limit of 10, the chances are less than one in 250. Of course, this probability will rise when the entity mentions are not evenly distributed. However, representations that are computed by querying for groups of predicates that are JEPD (e.g. curved/straight, size) do lead to more evenly distributed entity mentions since one predicate out of the group must hold for any entity (or combination of entities).

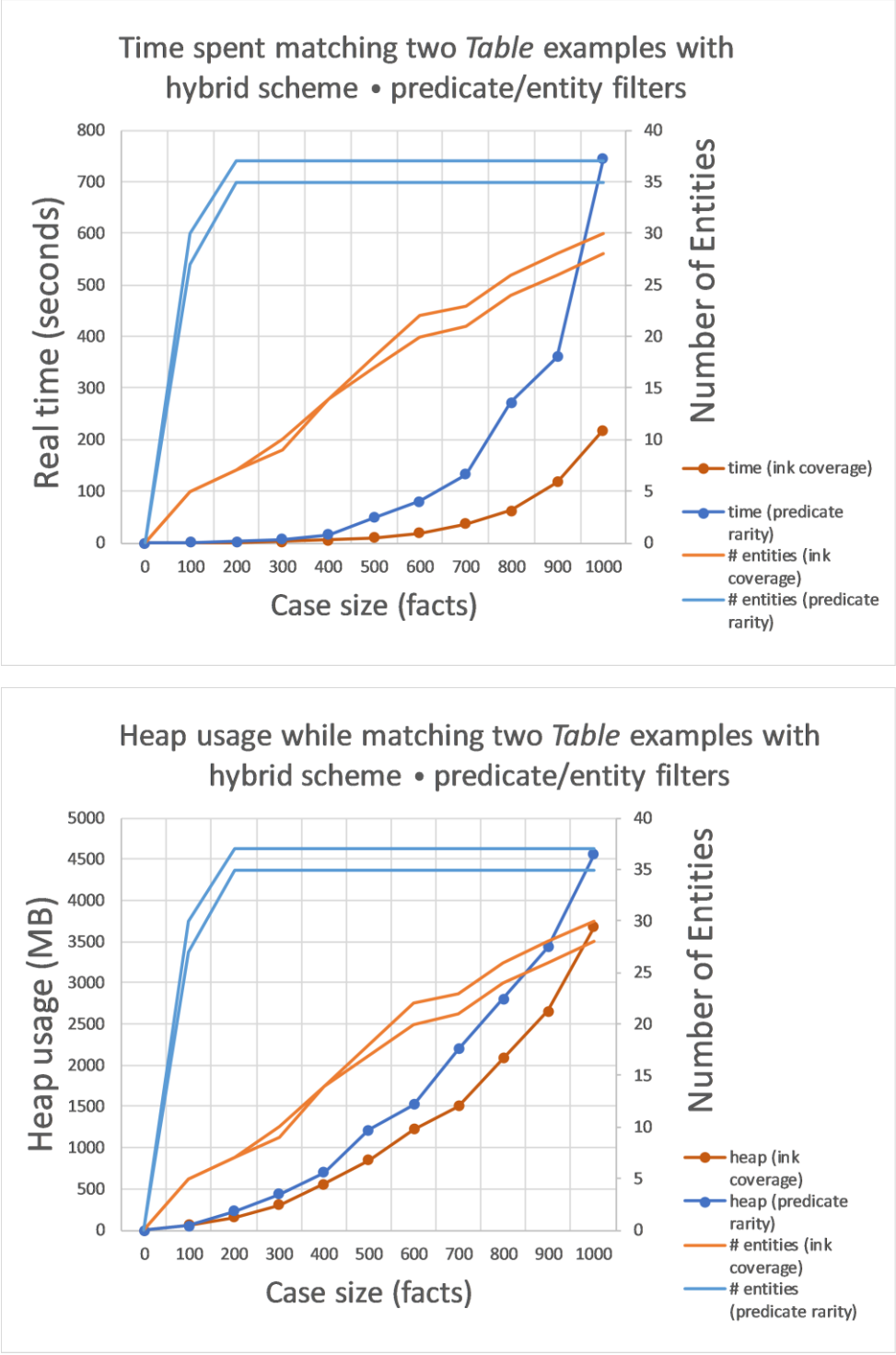


Figure 3.14: The time (top) and heap (bottom) spent while matching the two tables from Figure 3.11, encoded using the entity-based ink coverage filter (orange) and the predicate rarity filter (blue). The number of entities in the base and target for each scheme are also plotted.

guaranteed to mention all entities for commonly used case sizes, which ameliorates the *what goes with nothing* problem by providing an opportunity for any entity correspondence from the unfiltered case to be in the filtered case. We test a predicate filtering scheme that is not necessarily independent of entity, but for which no correlation with certain types of entities is obvious to the author.

We test a predicate-based filtering scheme that repeatedly finds the rarest predicate in the case (choosing arbitrarily between predicates that tie for rarest) and pulls in the facts that use that predicate. This additionally resists the *what goes with nothing* problem by downgrading the most common predicates, reducing misleading overlap from common predicates.

It may be that the rarest relationships are disproportionately salient, and thus are disproportionately intended to signal a label. Furthermore, there is precedent for including anchoring relations – relatively rare higher order relations – in filtered encodings for SME, in a bid to let salient regions of the sketch drive the mapping (Lovett et al. 2007).

On the other hand, there is also reason to believe that noise is correlated with rare predicates since a sketched object may be intended to have certain regularities. For example, in instances of a concept that is intended to be drawn entirely with straight edges (e.g. *Envelope*) a mistakenly curved edge will stand out and disproportionately affect comparisons.

Figure 3.14 compares the ink coverage filter to the predicate rarity filter in terms of the resource consumption (time and memory) of an analogical match between the tables from Figure 3.11, encoded with hybrid encoding scheme. The predicate rarity filter brings in all entities from base and target by the time it hits 200 facts, skirting the *what goes with nothing* problem. The entity-based filter brings in entities more gradually but consumes less memory and far less time during analogical matching. The entity-based filter exceeds 1 second of run time by the time it

reaches 300 facts (2.1 seconds) and exceeds 10 seconds by 600 facts (18.3 seconds). The predicate-based filter exceeds 1 second of run time by 200 facts (2.1 seconds) and exceeds 10 seconds by 400 facts (16.4 seconds). Experiment 2 below compares the efficacy of these filters for recognition.

3.4 Experiment 1: Edges vs edge-cycles on Sun Up to Sun Down

This experiment, first reported in (McLure et al. 2011), compared edge-cycles to edges for classifying the Sun Up to Sun Down dataset (Section 2.3.4). The edge-based representations, from Lovett et al. (2006), used CogSketch's edge level vocabulary as described, extended with some extra edge-level relations to encode simple cycles, but also reduced by ignoring relationships between two internal edges – ones that weren't along the perimeter of the object. To be tractable even for complex sketches, which contained up to 600 facts using that encoding, they filtered down the cases using a strategy that factored in both entity preferences and predicate preferences. They found 175 facts to be a peak-performance case size while testing up to a size of 225 facts, suggesting that this filtered case size also would outperform an unfiltered version.

We compared to that an edge-cycle-based encoding that only included atomic edge cycles, perimeter edge-cycles, and edge-connected objects. Continuity edge-cycles were not used in this experiment, nor was the between relation. This representation was unfiltered. We hypothesized it would outperform the edge-based encoding, even though the edge-level representation had been partially optimized for case size and entity-preference (albeit with a slightly different experiment structure and analogical generalization approach).

This experiment used the SAGE-based Retrieve-NN classifier from Section 2.1.4. The edge-cycle encoding was compared to the edge encoding for classification efficacy.

We used a 10-fold cross-validation, with the 8 sketches generated by a participant constituting a fold. This resulted in 10 rounds of training and testing per condition, where the system does analogical learning over nine participants' sketches and categorizes those of the tenth. To explore how SAGE's assimilation threshold (inversely related to the amount of compression) affected performance, we ran this experiment across a range of values for this parameter, from 0 to 1. We tested at higher granularity for the range from 0.5 to 0.8, where SAGE has historically performed well on other classification tasks. Chance was 0.125.

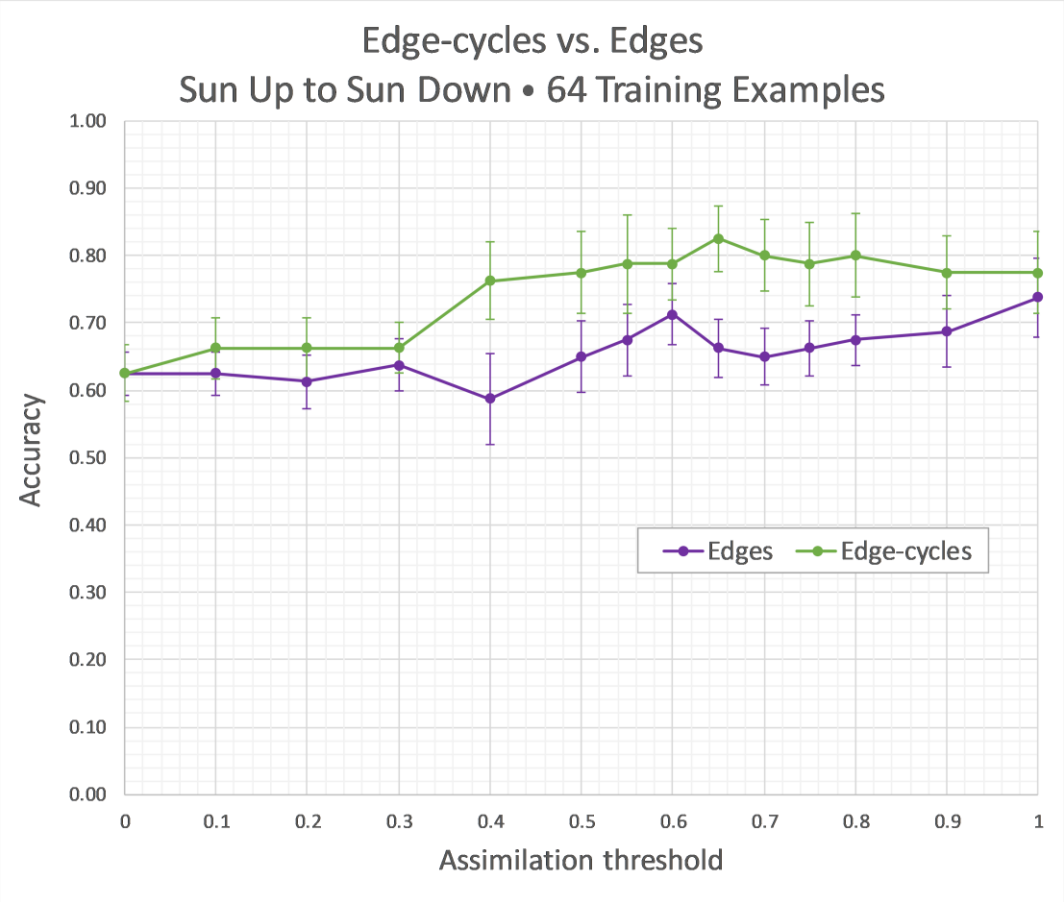


Figure 3.15: The performance of the Retrieve-NN classifier using edge-cycles (green) vs. edges (purple), over a range of SAGE assimilation thresholds.

3.4.1 Results

Figure 3.15 summarizes the results. Recall that the assimilation threshold (S) is the minimum similarity score (computed by SME) required between an example and another example (or generalization) for SAGE to combine them. With S = 0, SAGE always combines descriptions, and the two encodings achieved equal performance. For every other value, the cycle-level encoding outperformed the edge-level encoding. When $0.4 \leq S \leq 0.8$, with the exception of S = 0.6, the performance was significantly different ($p < 0.05$, one-tailed paired t-

test). The difference between the maximum accuracies achieved in the two conditions, which was at S = 0.65 for the cycle-based encoding and S = 1.0 for the edge-based encoding, was marginally significant at $p < 0.06$.

It is worth noting that the edge-cycle and edge schemes did not perform significantly differently at either end of the learning compression spectrum. The biggest differences were in an intermediate range of moderate learning compression rates. Edge-cycles were robust to compression down to an assimilation threshold of 0.4 and may have even slightly benefitted from some compression. Edge-based classification generally suffered at even at these moderate compression rates (compared to uncompressed).

Figure 3.16 shows the confusion matrices produced by the classifier in the cycle-based and edge-based conditions, with the assimilation threshold set to 1.0. With that high of an

Cycle-Based, *S* = 1.0

Predicted

Actual	Brick	Oven	Fridge	House	Fireplace	Cup	Bucket	Cylinder
Brick	10	0	0	0	0	0	0	0
Oven	1	9	0	0	0	0	0	0
Fridge	0	0	8	1	0	1	0	0
House	0	0	0	10	0	0	0	0
Fireplace	0	0	0	2	7	1	0	0
Cup	0	0	0	1	0	6	3	0
Bucket	0	0	0	0	0	1	7	2
Cylinder	0	0	0	0	0	2	3	5

Edge-Based, *S* = 1.0

	Brick	Oven	Fridge	House	Fireplace	Cup	Bucket	Cylinder
Brick	9	0	1	0	0	0	0	0
Oven	1	6	1	2	0	0	0	0
Fridge	1	0	8	0	1	0	0	0
House	0	2	0	8	0	0	0	0
Fireplace	0	0	0	0	10	0	0	0
Cup	0	0	0	0	0	6	3	1
Bucket	1	0	0	0	0	3	5	1
Cylinder	0	0	0	0	0	2	1	7

Figure 3.16: Confusion matrices for the edge-cycle-based (left) and edge-based (right) encodings for an assimilation threshold of 1 (no compression).

assimilation threshold, SAGE never generalizes, and hence classification consists of finding the closest (ungeneralized) example. This factors out analogical generalization, so that we can focus on properties of the representations produced. The confusions matrices showed that cups, cylinders, and buckets were confusable in both conditions, but slightly more so in the edge conditions. Looking at edges makes it harder to recognize the bricks, houses, buckets and especially ovens, but makes it much easier to recognize fireplaces. Curiously, the edge confusion matrix shows a more even distribution of guesses, ranging from 8 to 12 for a given label. The edge-cycle condition showed a range of 7 to 14 guesses for a given label. It was very ready to call things houses, and very reluctant to call anything a fireplace or a cylinder.

3.5 Experiment 2: Perceptual organization and filtering effects on classifying the Berlin25 dataset

In this experiment we used a classification task to look at the efficacy of 18 filtered encoding schemes that vary along three dimensions, shown in Figure 3.17.

The first dimension is the perceptual organization method used: (1) A cycle-based encoding that included atomic, perimeter, and continuity edge-cycles, (2) an edge-based encoding that included super edges, and (3) an encoding that combines the edge-cycle representations from 1 and the edge-based representations from 2.

The second dimension was the filtering method. There were two types of filtering used on each perceptual organization method, predicate-based filtering and entity-based filtering. For edges, the ink-coverage measure was used to filter entities. This measure was also used to filter the hybrid perceptual organization scheme with edges and edge-cycles. For the pure edge-cycle

organization scheme, entities were filtered using area coverage, since the ink-coverage measure was unavailable due to the lack of edges.

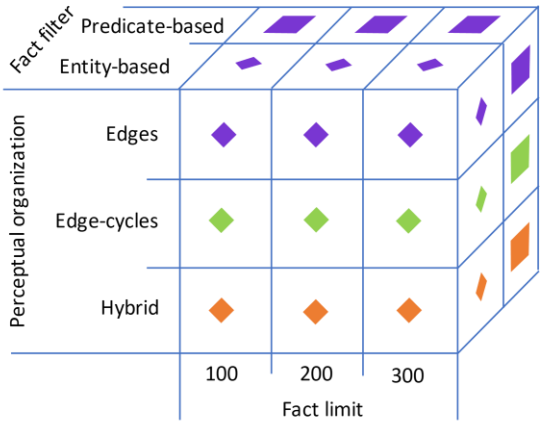
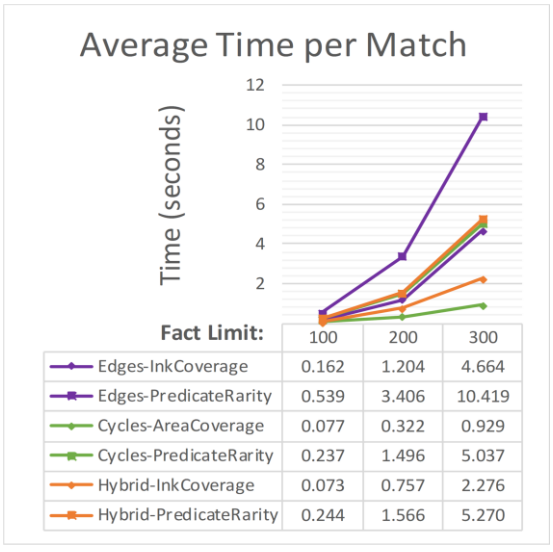
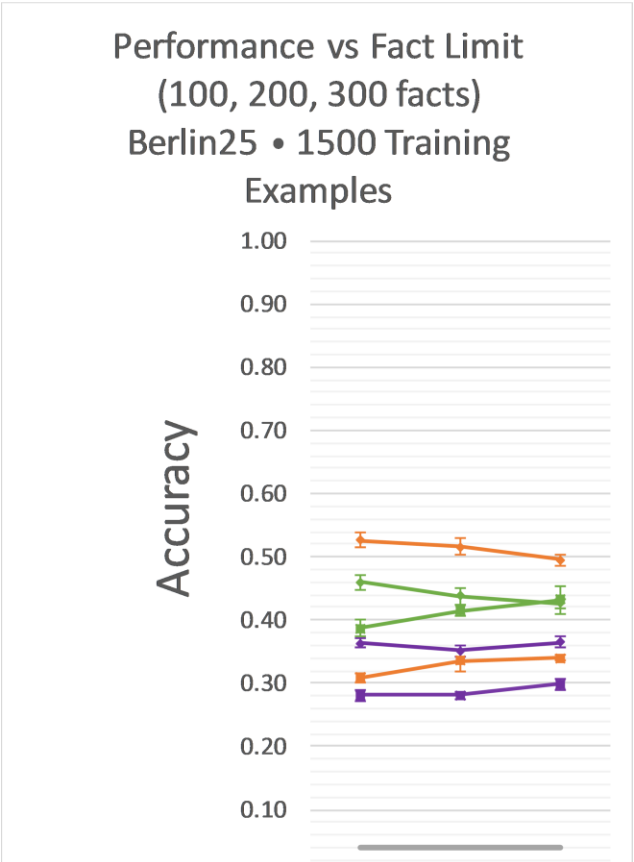


Figure 3.17: The grid of conditions in Experiment 2.



Fact Limit:	100	200	300
Edges-InkCoverage	0.162	1.204	4.664
Edges-PredicateRarity	0.539	3.406	10.419
Cycles-AreaCoverage	0.077	0.322	0.929
Cycles-PredicateRarity	0.237	1.496	5.037
Hybrid-InkCoverage	0.073	0.757	2.276
Hybrid-PredicateRarity	0.244	1.566	5.270

Figure 3.18: The average time taken to perform matches (i.e. SME runtime) when running on the Berlin25 dataset with 1500 training examples.



Fact limit:	100	200	300
Edges-InkCoverage	0.36	0.35	0.37
Edges-PredicateRarity	0.28	0.28	0.30
Cycles-AreaCoverage	0.46	0.44	0.43
Cycles-PredicateRarity	0.39	0.42	0.43
Hybrid-InkCoverage	0.53	0.52	0.50
Hybrid-IPredicateRarity	0.31	0.34	0.34
Chance	0.04	0.04	0.04

Figure 3.19: The performance of a similarity-based classifier Retrieve-KNNW over a matrix of different encoding schemes generated by crossing case size-limit (100/200/300) with filtering strategy (entity-based/predicate-based) with level of perceptual organization (edges/edge-cycles/both).

The third dimension that was manipulated was the size of the filter. We tested limits of 100, 200, and 300 facts per case.

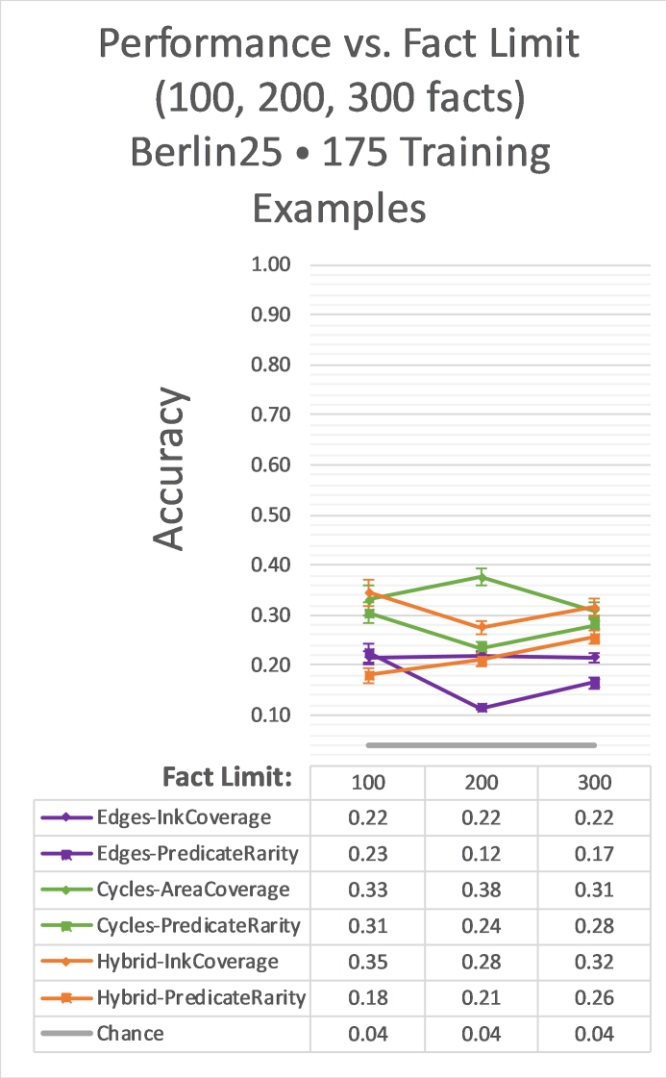
The task was to classify the sketches in the Berlin25 dataset (Section 2.3.1). We look at performance on the whole dataset (80 examples per concept of 25 concepts), as well as performance on 10% of the dataset (8 examples per concept, randomly chosen).

A 4-fold cross-validation structure was used for the whole dataset. In other words, for each concept, the 80 examples were randomly split into 4 folds of 20 each. This resulted in a training set, per trial, of 1500 examples (60 examples per concept) and a testing set of 500 examples (20 per concept). For the 10% dataset, an 8-fold cross-validation structure was used. Thus, there were 175 examples (7 per concept).

The Retrieve-KNNW classifier from Section 2.1.3 (sans SAGE learning) was used, with K = 9. SAGE was not used here because of the amount of time it would have taken to run all 18 conditions, especially those with 300 facts per case.

3.5.1 Results

On the full Berlin25 dataset of 25 concepts and 60 training examples per concept, the hybrid perceptual organization method (edges mixed with edge-cycles) with the entity-based filter outperformed all the other encodings handily Figure 3.19. Edge-cycles significantly outperformed edges. Predicate rarity performed universally worse than entity-based filtering, except for the largest fact limit and the edge-cycle (alone) encoding, where they converge. With predicate-based filtering, the hybrid scheme performed significantly worse than the edge-cycle scheme.



Fact Limit:	100	200	300
Edges-InkCoverage	0.22	0.22	0.22
Edges-PredicateRarity	0.23	0.12	0.17
Cycles-AreaCoverage	0.33	0.38	0.31
Cycles-PredicateRarity	0.31	0.24	0.28
Hybrid-InkCoverage	0.35	0.28	0.32
Hybrid-PredicateRarity	0.18	0.21	0.26
Chance	0.04	0.04	0.04

Figure 3.20: Classification accuracy on the miniature Berlin25 dataset, with 8 examples per concept. Since 8-fold cross-validation was used here, there were 7 examples per concept in the training set in a given trial.

For the lower number of training examples (Figure 3.20), edge-cycles tended to outperform edges and the hybrid scheme, although they were not significantly different in some conditions. The best performing condition for this amount of training was a fact limit of 200 on entity-filtered edge-cycles.

We did not end up seeing a common peak in the filter size, nor even a relationship between case size and performance that was common to all encoding schemes. Increasing case size had a mildly positive effect on performance for predicate-based filters and mildly negative for entity-based filters. The top two performing conditions are the 100- and 200-fact versions of the best organization scheme (hybrid) and the best filtering strategy (entity-based).

Predicate-based filtering led to far more time-intensive matches for all schemes and fact limits, consistently taking more than twice as long. Edge-cycles led to the least intensive matches, followed by the hybrid scheme.

Edge-cycles ultimately produced case libraries that were roughly half the size of their hybrid and edge counterparts, a reflection of the increased number of cases that did not reach the limit.

3.6 Discussion

In both experiments we saw edge-cycles significantly exceed the performance of edges in supporting the analogy-based classification of sketched objects. The results of Experiment 1 indicate that even atomic and perimeter edge-cycles, in the absence of continuity cycles, are a significant improvement over edges. They also suggest that edge-cycles are more resistant to the compression resulting from a lower assimilation threshold, although the amount of compression was not measured directly. Experiment 2 showed this relationship held after adding in continuity cycles to the edge-cycle encoding. It also shows that this relationship held over disparate training set sizes (1500 examples and 175 examples).

It is evident from the results of Experiment 2 that the entity-based filtering using visual efficiency significantly outperformed predicate-based filtering for most fact limits and perceptual organization schemes, especially with more training. With the larger training set size, the

difference was only insignificant at the largest case size for the sparsest perceptual organization, edge-cycles. This makes sense, because there is relatively little filtering happening in those conditions, leading to relatively little difference between in predicate- end entity-filtered cases. This convergence, together with the fact that predicate-based performance dropped as case size decreased while entity-based performance increased, suggest that, at least for edge-cycles, predicate-based filtering degraded performance while entity-based filtering improved performance (over unfiltered).

Experiment 2 also showed that at the larger training set size, with the superior entity-based filter, an encoding scheme that combined edges and edge-cycles significantly outperformed both edge-cycles alone and edges alone, with edge-cycles outperforming edges. However, on the smaller training set size, the relative efficacy of the hybrid scheme and the edge-cycle scheme was less clear, with the peak performance using edge-cycles filtered to 200 facts. Edges still underperformed. This may be due to the fact that edge-cycles are more stable and resistant to overfitting than edges. When we have lots of training examples to suppress the noise (because more similar training examples can make it into the weighted KNN), the hybrid scheme, which has the richest information available, pulls into the lead. It suggests that the ideal representations to use may shift as more examples are observed.

The lackluster effects of filtering on predicate rarity is likely due to an outsized focus on unusual details of minor elements in a sketch. Small artifacts resulting from segmentation errors over noise in the ink, especially edge-cycles, can take on interesting shapes because they do not abide by the intended regularities of the sketch. For example, if a glyph consisting of all right angles is segmented to include a small false-positive edge-cycle, that edge-cycle is not as likely to consist of right angles. These false positive cycles are also likely to be small, possibly

explaining why a tight entity-based filter would be effective in ignoring them. The predicate filter likely had a particularly negative impact on the MAC stage of MAC/FAC, which uses content-vector dot products to estimate similarity between cases. Content-vectors are histograms that encode the frequency of predicates. When filtering on predicate rarity, we remove the peaks in the histograms, systematically corrupting the signal used by MAC, which is used to discard the vast majority of relevant training examples. A lightweight addition to explore in future work would be to store the content vector for the unfiltered case in association with the filtered cases that derive from it, even after discarding the unfiltered case itself. MAC may benefit by synthesizing two stages of content vector comparisons – unfiltered and filtered.

For large training sets, there was an interaction between using the edge-cycle scheme vs. the hybrid scheme and using entity-based vs. predicate-based filtering. When using a predicate-based filter, edge-cycles outperformed the hybrid scheme. When using entity-based filtering, the reverse was true, with the hybrid scheme underperforming even the edge-based, entity-filtered combination. It is clear that there is as much to lose as there is to gain by adding edges to an edge-cycle representation, and that focusing on the largest edges can be beneficial while focusing on the most idiosyncratic edges can be harmfully distracting.

Interestingly, Figure 3.19 and Figure 3.18 and show an approximately inverse relationship between the performance of an encoding scheme and the average time it takes to match cases encoded with that scheme, holding case size constant. This suggests that the number of entities may be an important factor in how likely SME is to get derailed by too much detail. It may alternatively (or additionally) suggest that the matcher and performance benefitted from a higher

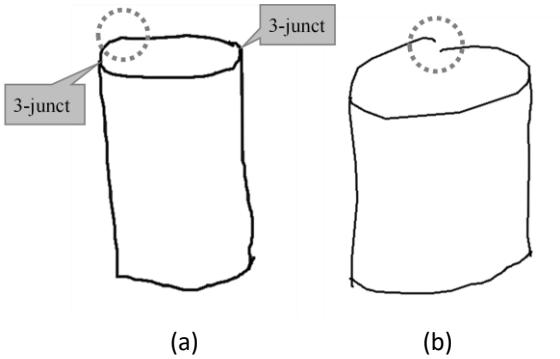


Figure 3.21: Sketches of cylinders whose edge segmentations were problematic. Cylinder (a) has an unintended junction resulting from a curvature discontinuity, and cylinder (b) has an unintended gap.

ratio of attributes to relations, since edge-cycles, as curvilinear polygons, can take on a wider variety of shapes. Edge representations are more dominated by relations, which may be swamping and misleading the matcher.

An analysis of the errors in Experiment 1 revealed that one large source of error in edge-level representations was curvature-based segmentation errors. Recall that digital ink generally contains artifacts that can make accurate segmentation difficult. Edges that connect end-to-end at 2-way junctions can be especially difficult because they are highly subjective. The detection of false positive junctions and omission of junctions can have a significant impact on the edge-level representation. For example, in Figure 3.21(a), the cylinder's top ink contour is split into two edges at the circled location, due to a change in curvature. This means that different entities are participating in the three-way junctions on either side of the top. Since SME requires mappings to be 1:1, this substantially reduces the similarity computed for the edge-level representation. The edge-cycle representation is robust relative to extra 2-way junctions because it abstracts the edges away to cycles, keeping the number of entities the same and potentially capturing differences like these in attributes and relations.

On the other hand, the edge-cycle encoding has its own set of problems with substantial topological imperfections. The cylinder in Figure 3.21(b) contains a gap in the top. Not closing this gap leads to an edge-cycle description with half of the atomic edge-cycles of the cylinder in Figure 3.21(a), changes the perimeter edge-cycle to no longer be concave, and adds an edge-

protrusion attribute. These are changes that radically reduce perceived similarity by the system, although people seem to "fill" these gaps reasonably easily. The edge-level representations are generally more stable when gaps occur, since the number of entities may or may not stay the same (gaps at corners vs on the middle of curves, like the one in Figure 3.21(a). Even when there is an extra or missing edge, it represents a smaller proportion of the total entities. This is especially true for small, simple sketched objects, which will have few edge-cycles.

As discussed in Section 2.3.1, sketches with textures can contain an explosive number of atomic edges. Edge-cycles do not alleviate the issue for most textures. In fact, for many textures, the edge-cycle representation is even more bloated than the edge representation. The house in Figure 2.10 (left), had 56 entities and 3,406 facts with the edge encoding. It had 108 entities and 8,528 facts with the edge-cycle encoding. In motivating our filtering strategies, we emphasized the benefit of filtering completely at the knowledge level. This allows the filter to be applied to cases in long term memory without accessing their original sketches and to hypothetical examples for which a sketch has never been observed, such as probabilistic generalizations. However, the downside is that the unfiltered case must be encoded in order to encode the filtered case. For sketches with textures, encoding the unfiltered case first can be impractical both in how long it takes to compute spatial relations (there are many pairs of neighboring edge-cycles and neighboring edges in a texture) and in how much space the case consumes in long-term memory. This suggests that encoding spatial relations should be preceded by texture detection to filter the textures' constituent perceptual entities completely. The technique in (McLure et al. 2015b), discussed in more detail in Section 6.2.2, finds textured regions before encoding. It reifies the texture regions instead of their member entities and assigns the texture regions attributes to reflect the geometric commonalities among their member

entities (e.g. major-axis orientation). Integrating this approach may help scale up to the full TU Berlin dataset, though additional handling is needed to detect textures that are not localized (e.g. a sketch with rain drops distributed throughout). More generally, intractable encodings could be avoided by imposing a limit on the number of perceptual entities after segmentation but before querying spatial relations.

In evaluating learning approaches in the next two chapters, when filtering examples, we use an ink-coverage filter over the hybrid encoding scheme with a fact limit of 100 – the best performing scheme on the larger dataset in Experiment 2.

4 Extending Analogical Generalization with Near-misses

Similarity can be deceptive. This chapter investigates approaches for detecting and leveraging highly similar pairs of positive and negative examples (near-misses) to improve on a similarity-based classifier commonly used by structure mapping approaches. This similarity-based classifier was introduced as *Retrieve-NN* in Section 2.1.4.

The techniques tested in this chapter and the next were designed with the following priorities:

1. *Data efficiency:* The ability to support recognition after few training examples.
2. *Fast classification judgments:* Speed in labeling an example.
3. *Disjunctive prototypes*: A set of visual prototypes for each category where every (observed) example of the category is sufficiently similar to at least one of the prototypes.
4. *Generative statistics*: Statistics for each category that capture relative prevalence of each prototype, as well as the relative prevalence of the visual properties of each prototype.
5. *Testable hypotheses*: Hypotheses about which properties of each prototype form boundaries for the category definition, i.e. those properties which, when not present in similar examples, are likely to disqualify those examples from belonging to the category.
6. *Explainable hypotheses:* Hypotheses that can be communicated to a person.
7. *Sublinear storage growth*: Long term memory requirements expand less than linearly with the amount of training data observed.

The three techniques put forth strike different balances between these competing priorities.

Why these priorities? Most were motivated in Chapter 1, but here we add generative statistics and testable hypotheses. Sketching software collaborators should be capable of recognizing *and generating* sketches. Sketch generation would be an asset in its own right – for example, when making creative contributions in design, teaching targeted concepts in tutoring, or communicating with the illiterate – but more subtly, sketch generation could facilitate an efficient refinement of visual categories for more sophisticated recognition by cooking up particularly informative, unobserved examples to ask a human about – a form of active learning called query synthesis. Active learning comes in different forms, many of which have in common the assumed existence of an oracle – a source that can be queried with an unlabeled example to elicit a label for it. For example, a learner may have a pool of unlabeled examples from which to choose the most informative one for the oracle to label. Or, in a streaming setup, an online learner, when shown each new example, must decide whether to ask for a label. These situations both involve judgments about examples that originate from some outside source. Query synthesis is a more sophisticated technique, in which the learner itself generates original examples it deems maximally informative, about which it then queries the oracle. We pointed out in Chapter 1 that sketches are relatively expensive for humans to produce. Combined with the wide space of things that can be sketched, this suggests that preexisting examples that probe a boundary at all, let alone optimally, may be hard to come by, and that query synthesis may be especially useful in sketch recognition. For this technique to be available in a structure mapping learning approach where the internal representations are relational and qualitative, we require a method for instantiating in ink an imagined (internal) representation of a novel informative example. We return to this endeavor in Section 7.3.6. Maintaining generative statistics for a category and its structured models, or prototypes, supports the instantiation of prototypical

instances of the category, from which local boundaries can be tested. The discriminative hypotheses learned by the approaches in this chapter and the next are highly testable because they are expressible in terms of elements in the structured prototype to which they pertain (as opposed to a similarity-based hypothesis).

We also pursue classification approaches whose long-term memory (LTM) requirements grow more and more slowly as examples are observed. The training processes for the classifiers evaluated in this section are built on the same foundation – SAGE – described in Section 2.1.4. SAGE is used to compress (cluster) training examples into a group of probabilistic models for each category. This is designed to scale to a range of concepts that a human could encounter better than attempting to remember every observed example, and more holistically representative than selectively retaining examples. Most of the experiments examine how accuracy changes with different levels of compression.

The clustering process in SAGE uses structural similarity as a signal for opportunities to cluster and compress examples. The statistics kept by SAGE for each model are the frequencies of every statement in each generalization – specifically, the frequency with which the examples assimilated into the generalization contained a statement that mapped onto that statement in the generalization. Note that this is not a full joint probability table for the statements in generalization – the number of frequencies kept is equal to the total number of all statements in all generalizations, for all categories (as opposed to all combinations of statements within every generalization, for all categories). This aspect of the approach is designed to scale.

4.1 Near-misses and category boundaries

Winston (1970) pointed out that learning to recognize a category via positive examples alone has intrinsic limitations. Even a large, representative collection of positive examples can only provide insight into the frequency and cooccurrence of properties, rather than their importance in determining membership. Even if the examples are chosen by a teacher, there is no way for the teacher to demonstrate that a specific property is critical in determining category membership. If the teacher hopes to convey which properties are essential or prohibitive for category membership, Winston argues that a *near-miss* should be provided – a negative example that differs from an observed positive example in very few ways.

The more similar the near-miss is to its positive counterpart, the fewer the alternative explanations for their differing labels. Winston framed these explanations as necessary conditions – properties that must or must not be present in an example to include it in a category. The properties were implemented as pointers to elements in a structured visual model for the category – MUST pointers for properties that block membership if absent, and MUST-NOT pointers for properties that prohibit membership if present. The learner prioritized and transitioned between these explanations by maintaining a current working model of each category along with a tree of alternative models that are consistent with all observed positive examples and near-misses.

For an example of a necessary condition crossing a concept boundary, assume that all sketched sharks are drawn with pointed dorsal fins. Removing the pointed dorsal fin from a shark drawing would disqualify it as a shark. It is a necessary but not a sufficient condition, since the presence of a pointed dorsal fin does not guarantee that the sketch is of a shark.

(Dolphins are often drawn with similar dorsal fins.) Alternatively, the near-miss may cross a boundary because one or more properties of the positive example that are different in the negative example are sufficient conditions for category membership. For example, assume that all sketches of palm-tree-looking things with coconuts hanging from their centers are drawings of palm trees. Adding hanging coconuts to a drawing that otherwise looks palm-tree-looking would be enough to qualify it as a sketch of a palm tree. This could be the key difference in a near-miss. Some conditions are both necessary and sufficient. Among zebra-like sketched objects (e.g. horse, mule… etc.), stripes may be a necessary and sufficient condition for membership in the zebra category. Still other properties are useful discriminators but are neither necessary nor sufficient. Among sketches of sharp instruments, swords tend to have more oblong blades than knives. They are more likely to have hilts and more likely to have a concave edge (resulting from a curved blade). None of these properties is true of all swords, nor untrue of all knives (and other non-sword sword-like instruments). For a near-miss in this case, it may be the cumulative effect of the differences or some subset of them that tip the scale from positive to negative. The techniques in this chapter are designed to learn necessary conditions, and those in the next chapter are designed to learn all of these types of discriminators. In any case, the nearer the miss, the more a learner is able to use the pair to home in on important differences, i.e. the narrower the set of resulting hypotheses about the boundary.

The examples provided have all been qualified to apply within a group of similar objects, e.g. "palm-tree-looking things." Universal necessary and/or sufficient conditions, which can be conveyed in a rule, are common in mathematics and law, but are elusive in sketch recognition if they are constrained to refer to purely spatial parts. Knowing that a shark must have a dorsal fin is not helpful for recognizing sharks when the inputs are spatial descriptions of ink unless some

subroutine can detect dorsal fins in spatial descriptions. Otherwise, the critical property must be framed spatially. But the ink that comprises a dorsal fin, like the ink that comprises a coconut in a palm tree, is often simple – three edges or fewer – to the point where its spatial description would be ubiquitous in many sketched objects. Moreover, the necessary configuration of the edges in the dorsal fin may seem unconstrained in isolation but may in reality depend on the pose of the shark and the perspective of the sketch. To be useful, a universal rule about a part must also capture enough of its context – the spatial properties of the rest of the sketched object's ink.

The discriminative techniques below do not attempt to learn sufficiently refined, globally applicable critical properties, nor do they eschew looking for critical properties altogether. Instead they learn locally applicable critical properties and use analogy to determine when and where to apply them. Consider the chairs in Figure 4.1. Coming up with a strict definition that

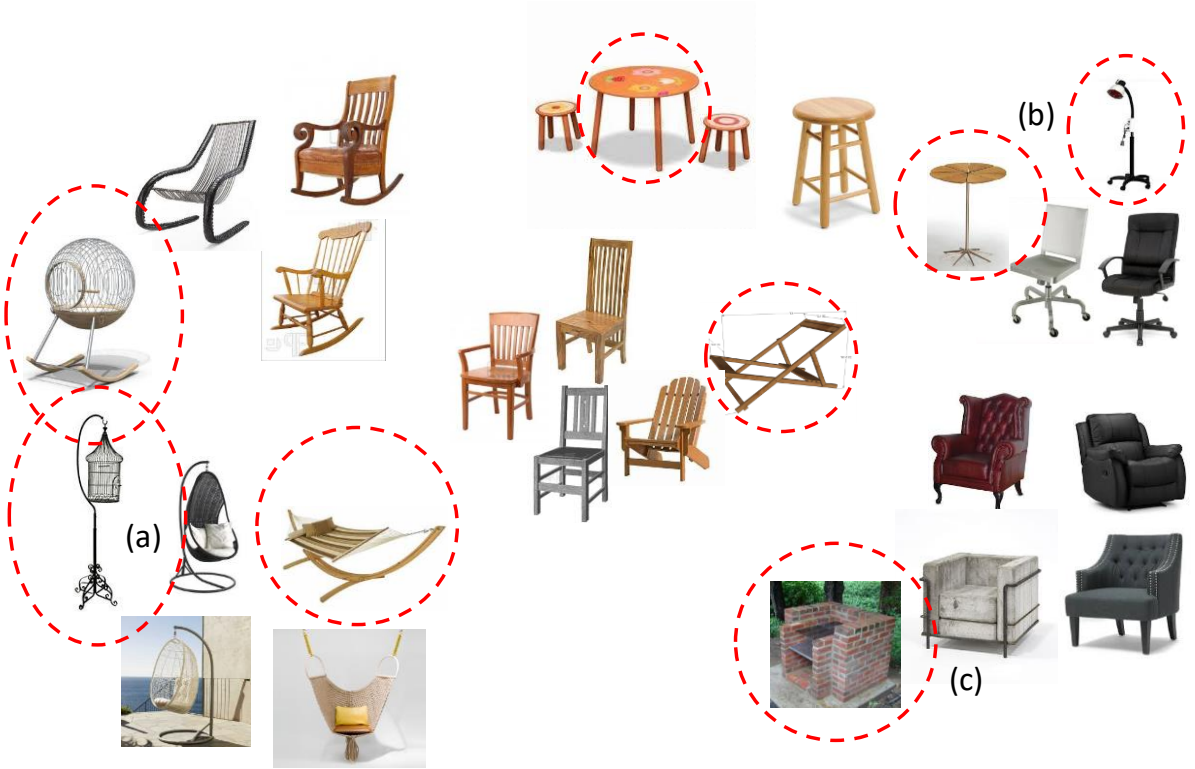


Figure 4.1: Some chairs and non-chairs (circled), roughly clustered according to similarity.

covers all positive examples and no negative examples is notoriously hard for chairs. However, notice that classifying chairs based on similarity to known prototypes is also problematic; some bird cages are very similar to some hanging chairs (a), some tables are very similar to some desk chairs (b), and some backyard grills are very similar to some arm chairs (c). Local comparisons with these near-misses may suggest criteria, e.g. for hanging chairs, *the hanging part must be below waist-level*, and *the hanging part must have a round concavity*. Notice that these criteria cannot easily generalize to desk chairs, because it isn't clear a priori what *the hanging part* would correspond to in a desk chair – they are not alignable. However, if later comparisons with desk chairs were to produce comparable criteria (e.g. *the horizontal part in the center must be below waist level*), then the learned criteria themselves could suggest a partial mapping between hanging chairs and desk chairs (e.g. *the hanging part of a hanging chair corresponds to the horizontal part in the center of a desk chair*).

A rule that is purely spatial can be simpler when it pertains to parts in a model and we only apply it locally – to similar objects that can be matched to the model. This is because the match itself can perform the job of determining which element(s) is/are the one(s) that the local rule applies to – a job that, in a globally applicable rule containing variables, requires additional

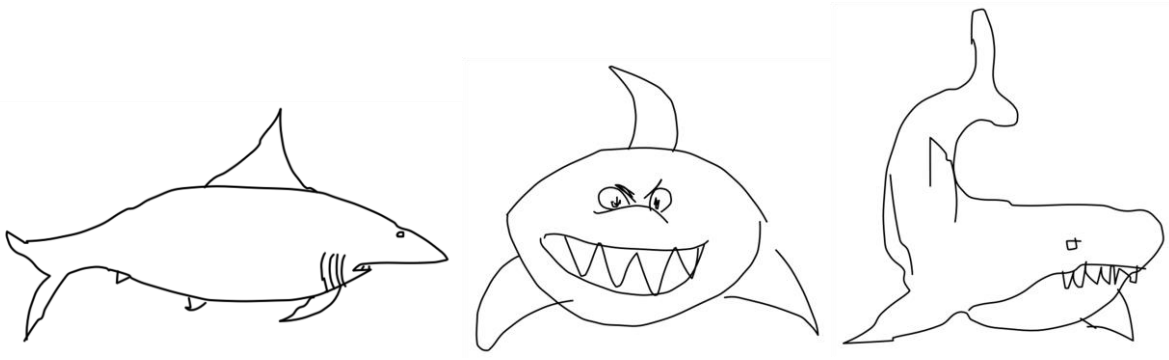


Figure 4.2: Three disparate sharks from (Eitz et al., 2012).

clauses in the rule to narrow down its scope to a particular element in a particular type of example in the category. When an object looks like a profile view of a fish (Figure 4.2, left), a dorsal fin would protrude upward from the top long edge of the elongated center portion (the body), taking a two or three-edged, pointed shape with an acute angle at the top. When an object looks like a fish viewed from the top (Figure 4.2, right), a dorsal fin would be enclosed within the large elongated (body) shape and while it would likely still consist of two or three edges, it could take on a wider variety of possible shapes and sizes because of the indirect viewing angle. It would likely have the same major axis as the body shape. The techniques below allow for these two distinct models of shark depictions to form, for rules pertaining to the model parts to be learned via near-misses and stored with the model, and for those rules to be applied selectively to sufficiently similar sketched objects, all automatically.

The extraction of critical properties from near-misses via analogical matching and storing them as references to parts in a model are core commonalities between Winston's near-misses and ours. The flexibility to spawn multiple alternative models (effectively learning a disjunctive definition by learning a set of necessary conditions for each model; Figure 4.3) – and the automatic discovery of near-misses (rather than receiving targeted near-misses from a teacher) are aspects where this work extends beyond Winston's. Another distinction is that we build on top of a prolific analogical matching algorithm from cognitive science in order to integrate with the ecosystem of work that assumes a matcher with the same properties.

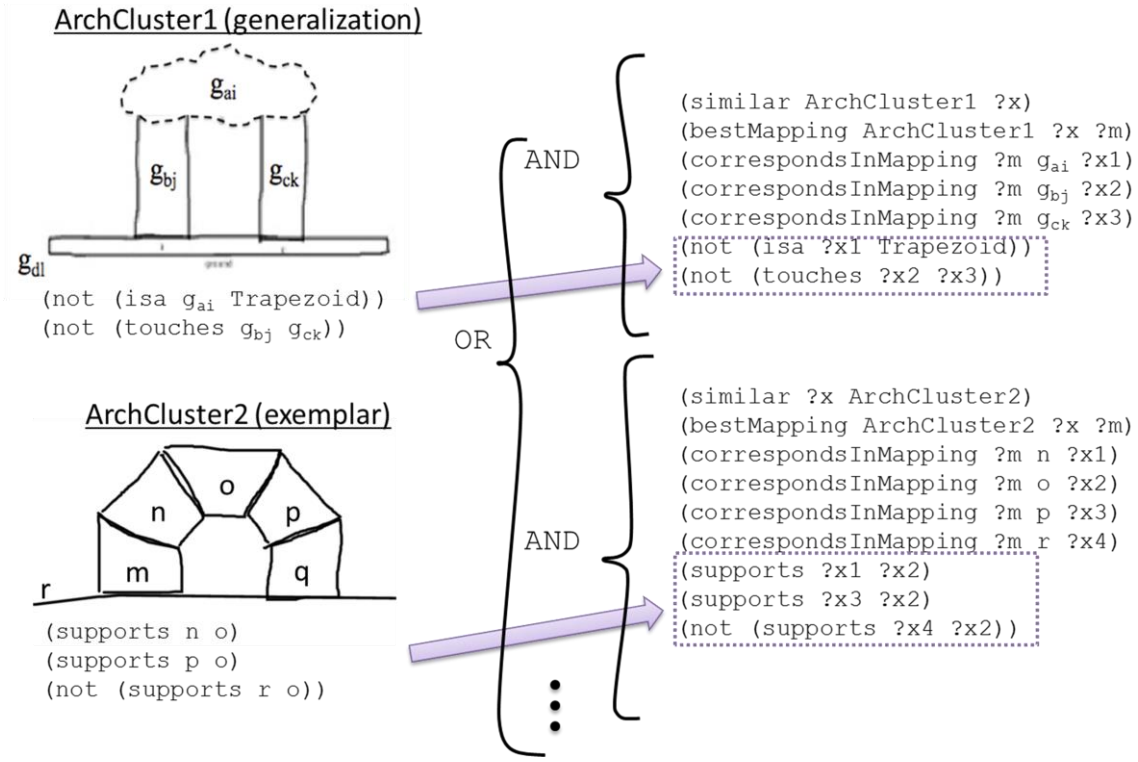


Figure 4.3: A view of how learning a set of necessary conditions for each of a category's prototypes can be seen as learning a disjunctive concept definition for the whole category.

Like Winston, the techniques describe in this chapter specifically look at the efficacy of learning necessary conditions for classification. A technique described in Chapter 5 uses a support vector machine (SVM) to find discriminative criteria during classification in a more general sense, without the semantics of necessity or sufficiency. In Section 7.3.3, we return to sufficient conditions and propose an avenue of future work in capturing them.

4.1 Learning necessary conditions via near-misses

Here we describe a learning algorithm published in Mclure et al. (2015a). It detected near-misses as it trained on labeled examples via similarity-based retrieval. It leveraged the near-misses to hypothesize necessary conditions for category membership via analogical comparison.

It used analogical generalization to refine its set of hypotheses. The efficacy of these necessary condition hypotheses in assisting similarity-based classification was demonstrated in a classification task, presented below in Experiment 3.

As mentioned, we used datasets consisting of mutually exclusive categories. Every example was labeled as at most one category, but the technique described is also applicable to examples with multiple labels. Every example was assumed to be a negative example of any category for which it was not assigned a positive label.

In this approach, the discovery of discriminative properties occurred during training. Upon observing an example of a category, MAC/FAC was used, with the new example as the probe, to retrieve similar examples from a case library containing all previously observed examples. Any retrieved example whose label(s) did completely match the new example – if its similarity score was above a threshold similarity score (the *near-miss threshold*) – constituted a near-miss for any label assigned to one example in the pair and not the other. To avoid introducing a new parameter, in all near-miss-based conditions in the experiments below, the near-miss threshold was coupled with the assimilation threshold used for all SAGE generalization pools, and they are jointly referred to as the *similarity threshold*. Near-misses that could be explained away with existing hypotheses associated with the positive example were not processed any further to extract new hypotheses.

The nearness of a near-miss was quantified by the degree of structural similarity between the positive and negative examples. MAC/FAC is not guaranteed to find the most structurally similar example in memory because that computation isn't tractable for large case libraries, but its second stage does filter based on structural similarity, and the similarity score and mappings that it returns with each reminding are computed by SME. Therefore, in so far as structural

similarity is a good measure of the nearness of a near-miss – a central premise in this approach – using MAC/FAC for near-miss detection was expected to be vulnerable to false negatives (near-misses in memory that aren't detected) but not false positives (detected near-miss pairs that aren't actually similar). Recall that MAC/FAC returns the SME match (one or more mappings) associated with each reminding. The top SME mapping returned for each near-miss pair was used to extract hypotheses about critical properties, as explained next in the next section. Section 4.3 describes how these hypotheses were used to augment similarity-based classification, and Section 4.4 explains how analogical generalization was used to filter the set of hypotheses to include only those that were plausible necessary conditions.

4.2 Hypothesizing Category Criteria from Near-Miss Comparisons

The value in a near-miss lies in the differences between the positive and negative examples, since one or more of this relatively narrow set of difference may explain the difference in label. In a match produced by SME between a base and a target, the candidate inferences (CIs) capture some of the differences by projecting unshared structure from base to target (translated to pertain to elements in the target). The reverse candidate inferences (RCIs) capture other differences by projecting unshared structure from the target back onto the base. Notably, unmapped expressions (unshared structure) are only projected as CIs and RCIs if they mention some entity or expression that does participate in the mapping.

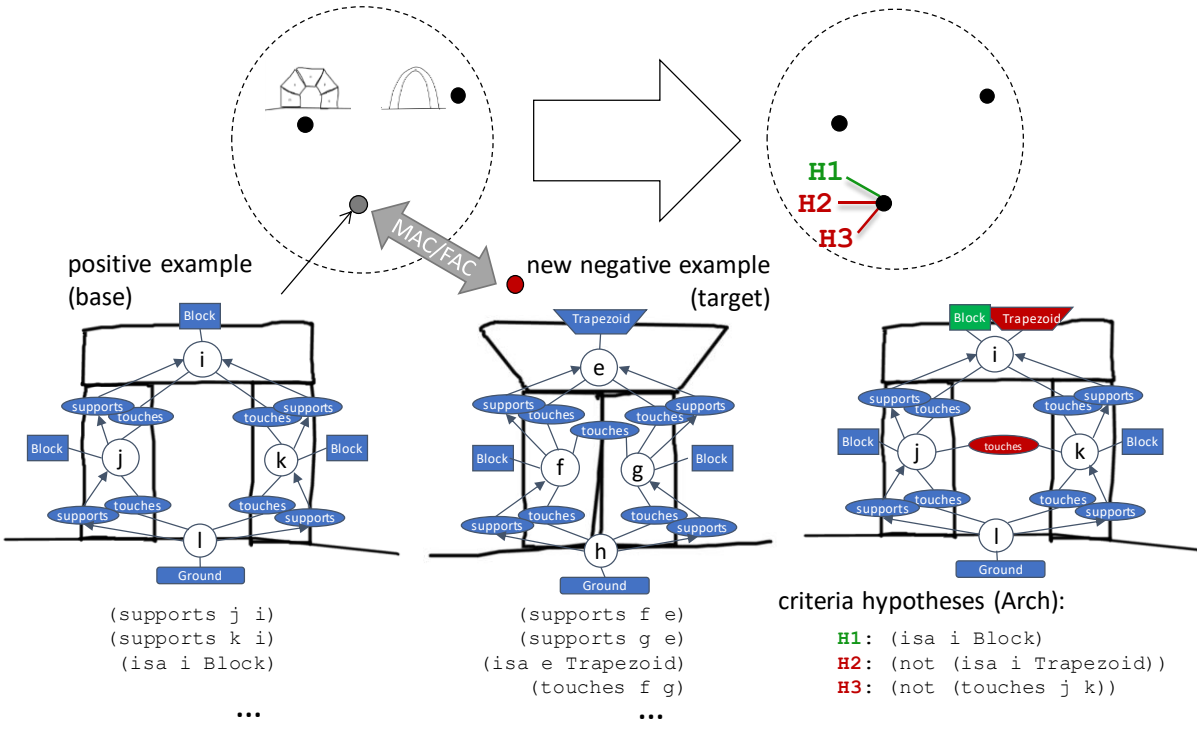


Figure 4.4: A negative example (middle) is observed, and MAC/FAC retrieves a similar positive example (left), resulting a near-miss that yields the single inclusion hypothesis (H1) and two exclusion hypotheses (H2, H3).

Consider the positive and negative examples of the concept Arch shown in Figure 4.4 (left and center, respectively). Thanks to alignable systems of relationships among entities in base and target, two of which are shown (the supports relationships), SME maps entity `i` to entity `e`, `j` to `f`, `k` to `g`, and `l` to `h`. If we take the positive example (left) as the base and the negative (middle) as the target, then there is one CI: `(isa e Block)`. There are two RCIs: `(isa i Trapezoid)` and `(touches j k)`.

In their raw form, the CIs and RCIs pertain to two separate sets of entities – CIs to target elements and RCIs to base elements. Next, we use the mapping to consolidate them to pertain to only one of the examples (base or target). Immediately consolidating the differences allows the

learner to discard the mapping between base and target without having to perform the match again later, which is important for tractability. It also prevents redundant hypotheses from accumulating. In this approach, we always consolidate by translating the differences to pertain to the positive example as hypothesized necessary conditions for category membership. This allows criteria testing to be parsimoniously incorporated into a similarity-based classification scheme (Section 4.3), since positive examples are the ones that potentially transmit label information when retrieved during classification. It also allows prototypes generated by analogical clustering and generalization to whittle down the set of criteria hypotheses (Section 4.4). Importantly, the positive/negative example may act as either the base/target or the target/base respectively. Below, we first explain how inferences projected from positive to negative are processed. The subsequent explanation of processing inferences from negative to positive spills into Section 4.2.1, due to some added complexity.

In the example in Figure 4.4, the positive example happens to be the base and the negative happens to be the target (a situation that would result from the negative example being observed and causing the positive example to be retrieved). Thus, in this case, the CI's, projected from positive to negative, pertain to entities in the negative example, and must be translated back to pertain to the positive example. This is accomplished by replacing all target entities embedded in each CI with their corresponding base entities, using the entity correspondences from the analogical mapping. For example, the CI `(isa e Block)` leads to the hypothesis that `(isa i Block)` is part of what qualifies the positive arch as an arch. In (Mclure et al. 2015a), this is referred to as an *inclusion hypothesis*, because it specifies a property that must be true. In Winston's original near-miss learner, this corresponds to a *MUST* pointer.

Although it is not the case in Figure 4.4, it is possible for CI's and RCI's to refer to entities that do not exist in the target/base respectively (the case onto which they have been projected). The process for producing near-miss hypotheses from these depends on the type of entity, and whether it is an inference projected from positive to negative or vice versa. An inference might mention a constant such as Zero, but this requires no translation since SME must have projected it without translation in the first place (from base to target for a CI or vice versa for an RCI). Otherwise, the non-corresponding entity must be a skolem, introduced in Section 2.1.2 as a hypothesized entity that, for example in a CI, does not exist in the target but was generated to correspond to some unmapped entity in the base.

Note that the existence of skolems in an inference is only possible if the inference also makes mention of some shared structure. This property of inferences in SME is crucial for near-miss learning, or else all superfluous unmapped structure in near-miss pairs would cause an explosion of criteria hypotheses containing skolems.

A skolem in an inference projected from positive to negative is simple to translate back to the positive example because it is named with a non-atomic term containing the (positive) entity from whence it came. For example, `(AnalogySkolemFn m)` translates to `m` in the base.

Inferences from negative to positive (RCIs in Figure 4.4) also require some processing. They already refer to entities in the positive example, but since they are projections of expressions in the negative example, their semantics are like censors, e.g. a condition matching `(touches j k)` should prohibit an example similar to the arch in Figure 4.4(left) from being an arch. As a formality, converting a hypothesized censor to a hypothesized necessary condition simply requires negating each one (if X is a censor for Y, then X implies ¬Y; it follows that Y implies

¬X, so ¬X is a necessary condition for Y). In Mclure et al. (2015a), these (negated) criteria are referred to as *exclusion hypotheses*. In Winston's near-miss learner, they correspond to *MUST-NOT* pointers. Processing for inferences from negative to positive containing skolems is explained in the next section.

4.2.1 Translating skolems into constrained open variables

Turning negative-to-positive inferences containing skolems into hypotheses about criteria requires special processing. For example, comparing the arch in Figure 4.4(left; base) with the non-arch in Figure 4.5(target) would produce six RCIs, projected from the relations colored purple in Figure 4.5 since these are the unmapped expressions that mention at least one entity that is mapped (as opposed to the attributes colored orange, which are unmapped expressions that exclusively refer to unmapped entities, thus producing no RCIs). All of the RCIs in this case contain a skolemized entity because all of the purple expressions refer to an unmapped entity w. One such RCI is `(touches j (AnalogySkolemFn w))`. Because we require that

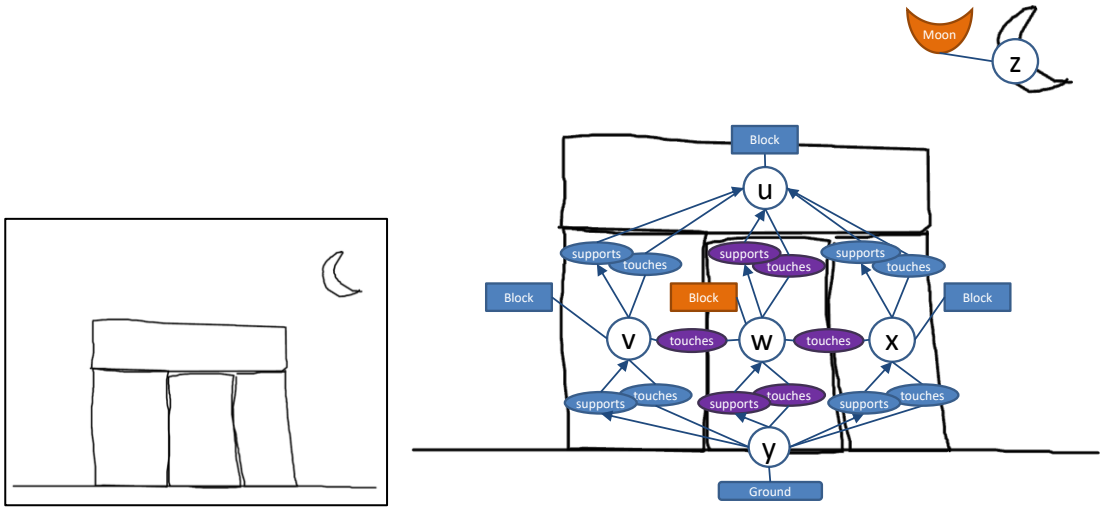


Figure 4.5: A non-arch with an extra entity that leads to criteria containing variables when compared to Figure 4.3 (left).

hypotheses refer only to entities in the positive example, and there is no entity in that example that corresponds to `w`, we replace the skolem `(AnalogySkolemFn w)` with a variable `?skolem`. This will allow us to check whether the criterion, projected onto an unlabeled example, is satisfied by some entity. But we can go further, based by the one-to-one correspondence constraint of structure mapping, by recognizing that the unmapped entity in the negative example that was meant to bind to the variable (`w`) was found not to correspond to any of the mapped positive entities. Therefore, we explicitly constrain the variable so that it cannot bind to any entity from the positive example that entered into an entity correspondence in the near-miss mapping. As with other exclusion hypotheses, a negation is added. When the resulting criterion is mapped onto an unlabeled example, the positive entities mentioned in the criterion will be translated according to the mapping, so the constraints will be projected to narrow the variable bindings down to extra entities, seeing as how the property came from something extra in the first place. The resulting hypothesis in our example roughly captures the necessary condition "There cannot be something *extra* that touches j," and looks like this:

```
(not (and (different i ?skolem)
     (different j ?skolem)
     (different k ?skolem)
     (different l ?skolem)
     (touches j ?skolem)))
```

Thus, skolems come closer to producing the kind of variable-based, logically quantified hypotheses that ILP (Section 6.1.2) deals in, but they are still constrained by the analogy from whence they came.

4.3 Criteria Testing using Structure Mapping: The Near-misses approach

Recall the Retrieve-NN approach discussed in Section 2.1.3, in which MAC/FAC is used to classify an unlabeled example by performing similarity-based retrieval over a case library of labeled examples. (For now, set aside the extension of Retrieve-NN to use SAGE in Section 2.1.4.) The most similar reminding retrieved by MAC/FAC is used to infer label information. For example, if an unlabeled example has caused a known arch to be retrieved then Arch may be a good label. We can extend this model to factor in the criteria hypotheses described in the previous section.

After training with near-miss comparisons, every case in the case library that is a positive example of some concept potentially has learned criteria hypotheses about that concept that refer to elements in the case. Retrieving cases from this library using MAC/FAC with an unlabeled example as the probe leaves us with an SME match between each reminding and the unlabeled example. For each reminding-match pair, for each concept for which the reminding is a positive example, we can project the hypothesized criteria for that concept associated with that reminding onto the unlabeled example for testing using the top mapping in the match. Hypotheses are translated using the entity correspondences in the mapping, simply replacing labeled example entities with their unlabeled example counterparts. The hypothesized criteria are queried within the unlabeled example to see if they hold.

How should satisfied or unsatisfied criteria affect classification? This approach hypothesizes that these criteria are necessary conditions, so we let unsatisfied criteria block remindings from transmitting label information. The remindings are tested in order of decreasing similarity until one is found for which no criteria are unsatisfied. MAC/FAC is called repeatedly if no

unblocked remindings are found – up to 3 times – at which point the remindings so far are evaluated for criteria support. Support is determined by subtracting the weight of the unsatisfied criteria from the weight of satisfied criteria. In the formulation described so far, in which remindings are simply labeled examples, every hypothesis has a weight of 1. Retrieved examples without labels, having no associated hypotheses, have neutral support of 0.

4.4 Analogical Generalization for Hypothesis Grouping and Refinement

Recall from Section 2.1.4 that the Retrieve-NN and Retrieve-KNNW classification methods can be applied to a union of generalization pools to retrieve prototypes. Like examples, prototypes can also have associated near-miss hypotheses. When positive examples are assimilated into a prototype with an analogical mapping, their associated criteria hypotheses may

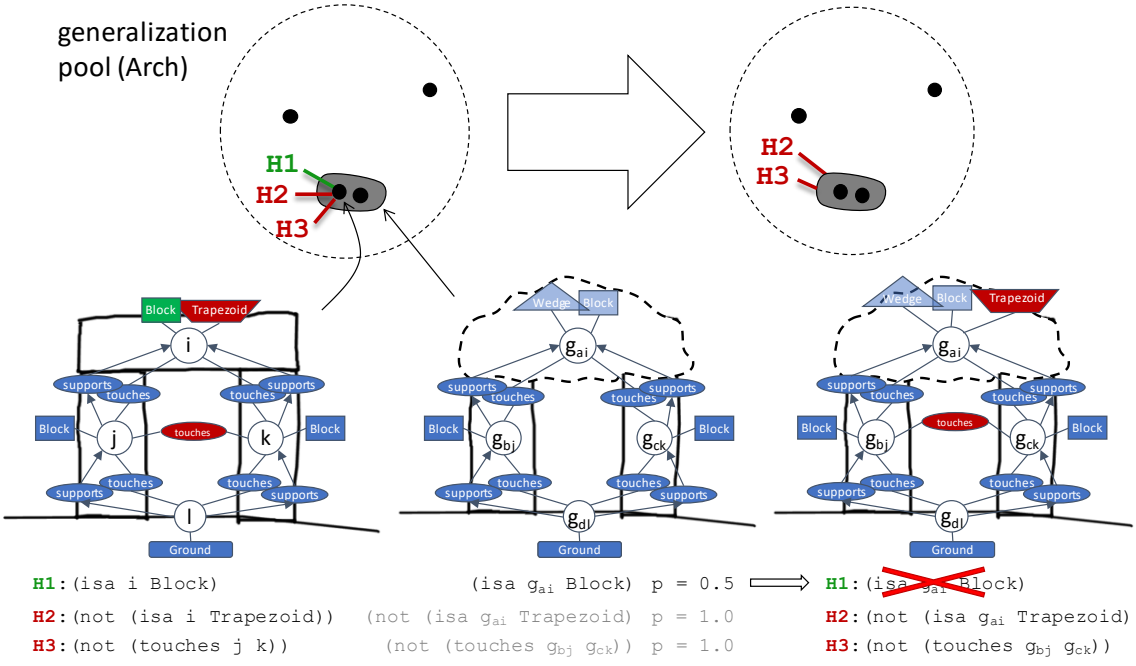


Figure 4.6: When a positive example (left) is assimilated into a generalization (center), the criteria pertaining to the example are generalized to pertain to the generalization (right), and are subject to pruning.

be projected onto the prototype via that mapping. Recall that the probabilities associated with expressions in a SAGE generalization reflect the frequency with which the assimilated examples contained expressions that mapped onto them. Here we infer that any expression with probability < 1.0 must not be a necessary condition because its analog did not hold in every similar positive example. Subsequently the hypothesis is pruned. The probability associated with the criterion in an exclusion hypothesis, $p(\neg X)$, is assumed to be equal to $1 - p(X)$, as in the case of H2 and H3 in Figure 4.6.

As a prototype assimilates more positive examples, it may inherit some new hypotheses from each one – any hypotheses that are not immediately pruned upon assimilation – but it may also refine its set of existing hypotheses as the new example causes more expression probabilities to drift away from the extremes, $p = 1$ and $p = 0$. Hypotheses that have survived more generalization are treated as more trustworthy, since differences resulting from noise or chance face probabilities that regress to a mean. Therefore, when computing criteria support during testing (per the previous section), criteria hypotheses associated with prototypes are given weight equal to the number of examples that have been assimilated into the prototype (ungeneralized examples still have weight 1).

But how does near-miss discovery change when analogical generalization is incorporated into training? Should the prototypes themselves enter into near-miss comparisons? Should positive examples be discarded after having been assimilated? This work explores two alternatives.

4.4.1 Refined-near-misses

The first method for incorporating analogical generalization into the near-misses approach, dubbed *Refined-near-misses*, was the method described and tested under the name ALIGN in (Mclure et al. 2015a). This technique keeps the example-based long-term memory of the Near-misses approach while adding analogical generalization as a parallel track for hypothesis refinement. Like in Near-misses, it keeps a case library for every training example, from which MAC/FAC, given a new training example as a probe, retrieves similar training examples and creates near-miss pairs from those with different labels from the probe. In parallel, each new training example is added to the generalization pool(s) corresponding to its positive labels(s). When near-miss hypotheses form for the probe and for the retrieval, they are immediately transferred to the generalization, if any, that each has been assimilated into. When new generalizations form between an ungeneralized example and the first example to merge with it, the hypotheses associated with the ungeneralized example transfer to the generalization as well. After being transferred, the old versions of the hypotheses, pertaining to the example, are discarded. At testing time, as each unlabeled example comes in, it is used to retrieve from the case library of examples. The associated hypotheses, normally projected from the retrieved example onto the unlabeled example for testing, must first be translated to the retrieved example from its associated generalization.

Importantly, translating hypotheses from assimilated example to generalization and back – a process needed even after the initial assimilation at times during training and testing, does not require keeping or regenerating a full analogical match between the assimilated example and the generalization. It does, however, require that an *entity history* be kept for every generalized

entity – a list of pairs associating each assimilated case with the entity in the assimilated case that corresponded to that generalized entity, e.g. {$Arch_1$, $Entity_1$}, {$Arch_2$, $Entity_2$}, …{$Arch_n$, $Entity_n$}.

As devised, the long-term memory requirements of Refined-near-misses scale super-linearly with the number of examples observed. Any generalization that may consolidate and prune hypotheses constitutes an additional case that is at least as large, in terms of the number of assertions, as the largest of the cases that have been assimilated into it. Even without storing near-miss hypotheses and generalizations, keeping every training example around in long term memory (which consumes space linear to the number of examples observed) is intractable for scaling up to a real-world collection of categories. Nonetheless, Refined-near-misses allows us to evaluate the added benefit of analogical generalization for hypothesis refinement. This is valuable because of potential avenues for modifying the Refined-near-misses method to be more tractable – specifically, techniques for selecting which (assimilated) training examples should be kept around for more productive, perhaps more diverse retrievals during near-miss discovery or classification.

4.4.2 Prototype-near-misses

The second method for incorporating analogical generalization into the Near-misses approach, dubbed *Prototype-near-misses*, is a simpler and more tractable extension. This method does away with the case library for all training examples and only maintains the generalization pools (one for every label, including one for the *null* label to store examples without any labels). The union of the generalization pools, which acts like a case library, is used for retrieving near-misses during training and labeled prototypes during classification. The

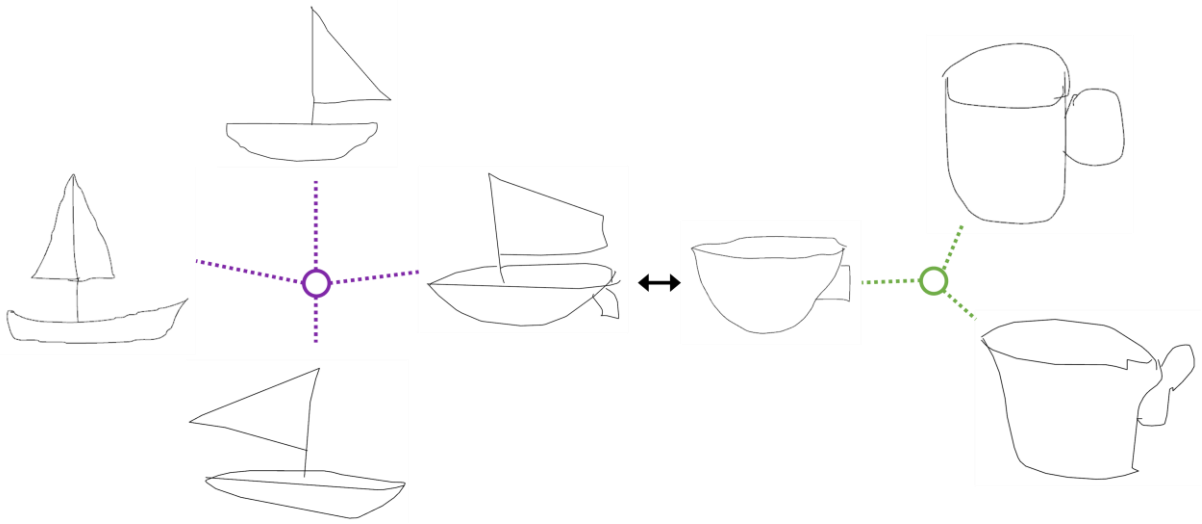


Figure 4.7: Sailboats and teacups illustrating a potential pitfall of Prototype-near-misses: lost near-misses.

incoming (ungeneralized) training example is still the probe for detecting near-misses, even if it is immediately assimilated. Since a generalization is a type of case, it is straightforward to apply the same near-miss extraction process between the training example and a generalization that was applied between two training examples. Most of the hypothesis translation needed in the training phase of Refined-near-misses is obviated, except for generalizing hypotheses that pertain to an ungeneralized example if and when it gets assimilated into a generalization. As in the Refined-near-misses method, a generalization's hypotheses are always pruned when the expression matching the content of the hypothesized criterion drops below p = 1 in the generalization. In the testing phase, hypotheses are translated directly from the (retrieved) prototypes onto the unlabeled example.

The Prototype-near-misses approach is meant to scale more realistically than Refined-near-misses because every training case isn't kept around. Prototype-near-misses is truly an online learning algorithm, updating its model as each new training example is observed, and discarding

the example (though the example has the potential to be kept as the seed for a new prototype). Refined-near-misses, on the other hand, has every previous training example available during training and every training example available during testing.

There are some functional differences that result from discarding assimilated examples and retrieving over prototypes, as in Prototype-near-misses, and these differences introduce risk. For one thing, some of the assimilated examples might have been highly similar to training examples from other categories – fodder for nearer misses. For example, the *Sailboat* and *Teacup* in the center of Figure 4.7 are a very similar, and therefore revealing, near-miss pair. Both are similar enough to other examples with the same label (shown) to get generalized by SAGE. In refined near-misses this does not affect the near-miss, but in Prototype-near-misses, depending on the order of the training examples, the first of the two examples from the near-miss pair to be observed could be assimilated away into a generalization before the second is observed, making the near-miss unavailable. A near-miss with the generalization could be detected instead, but it won't be as near and there will be more errant hypotheses generated. A similar risk is the assimilated examples might have been the most similar training example for some testing example. Merging with less similar training examples could pull the resulting generalization away from the testing example, giving it less influence than the assimilated example. Essentially, we are losing some degree of granularity in discerning where the category boundary falls in the similarity space.

A second key difference is that in the Prototype-near-misses classifier, generalizations can enter into comparisons directly. This could affect the usefulness of the mappings that produce/apply near-miss hypotheses at training/testing time, respectively. A really tight near-miss comparison – a positive/negative pair that are different in very few ways – is likely to

generate a more trustworthy notion of "what goes with what" than an alternative in which one or both sides have been saddled with the less analogous properties of other assimilated examples.

We can also expect a difference in the balance between inclusion and exclusion hypotheses. In Refined-near-misses, generalizations may not enter into comparisons directly to produce hypotheses, but they do affect which of those hypotheses survive. This only happens on the positive side of the comparison, since the criteria are framed as necessary conditions. In Prototype-near-misses, generalizations may enter into near-miss comparisons on the positive *or negative* side. There are implications of near-miss comparisons with generalizations on the negative side. A generalization on the negative side of a near-miss will tend to yield fewer inclusion hypotheses. This is because inclusion hypotheses originate from positive→negative candidate inferences, and those can only result from properties that are absent on the negative side (p=0). These are rarer with a generalization on the negative side because then it must be a property that isn't true of *any* of the assimilated negative examples (as opposed to just one). For exclusion hypotheses, which originate from negative→positive candidate inferences, generalizations on the negative side will tend to produce more hypotheses, since there will be more structure (anything p>0) to project as inferences.

4.5 Experiment 3: Impact of near-misses on classification tasks

This experiment evaluated the Refined-near-misses and Prototype-near-misses classification approaches with a similarity-based classifier, Retrieve-NN (Section 2.1.3), on sketch-recognition tasks. The near-miss classifiers are an extension of Retrieve-NN, and all are equivalent at a similarity threshold (S) of 1.

The first task was to classify the Sketched Concepts 2010 dataset (Section 2.3.3). The second task was classifying the Freeciv Geography dataset (Section 2.3.2). A cross-validation structure was used – 4-fold for the 2010 dataset, 10-fold for the Freeciv Geography dataset – where the examples for each concept were spread evenly across the folds.

No filters were used on the input representations for either dataset. The total number of hypotheses found during training (for all concepts) were recorded. On the 2010 dataset, a fixed similarity threshold of 0.8 was used, and chance was at 21%. On the Freeciv dataset, results were gathered over 11 similarity thresholds ranging from 0 to 1, and chance accuracy was at 17%.

4.5.1 Results

Figure 4.8 shows the accuracy and number of valid hypotheses found by the Retrieve-NN, Refined-near-misses and Prototype-near-misses classifiers on the Sketched Concepts 2010 dataset. The Prototype-near-misses method significantly outperformed both others significantly at 75% accuracy ($p<0.01$), while Refined-near-misses achieved 62% accuracy, significantly outperforming the similarity-based Retrieve-NN classifier ($p<0.01$). The Prototype-near-misses

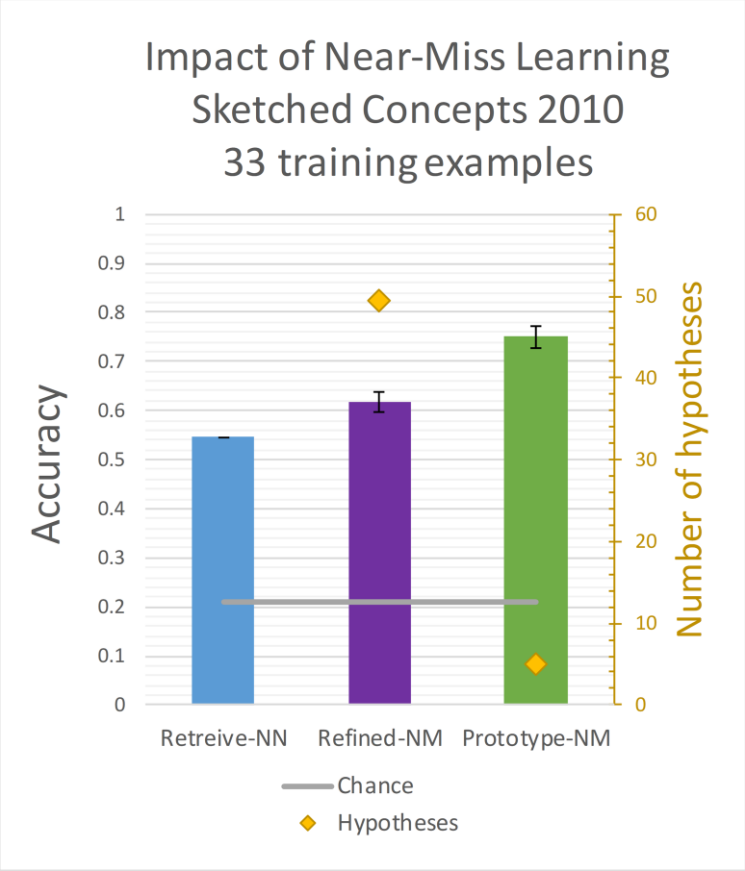
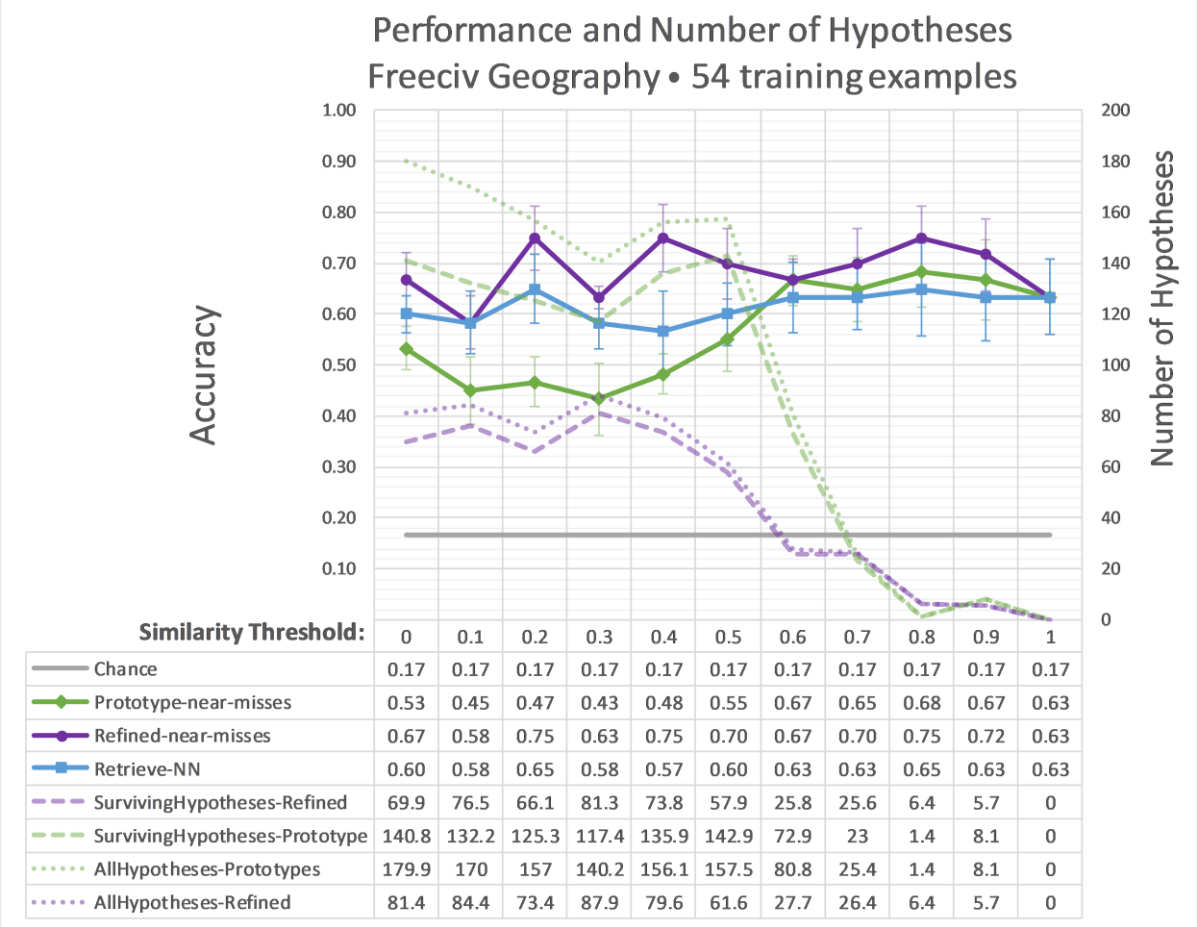


Figure 4.8: The accuracy and the average number of valid (surviving) hypotheses for Prototype-near-misses, Refined-near-misses and Retrieve-NN classifiers.

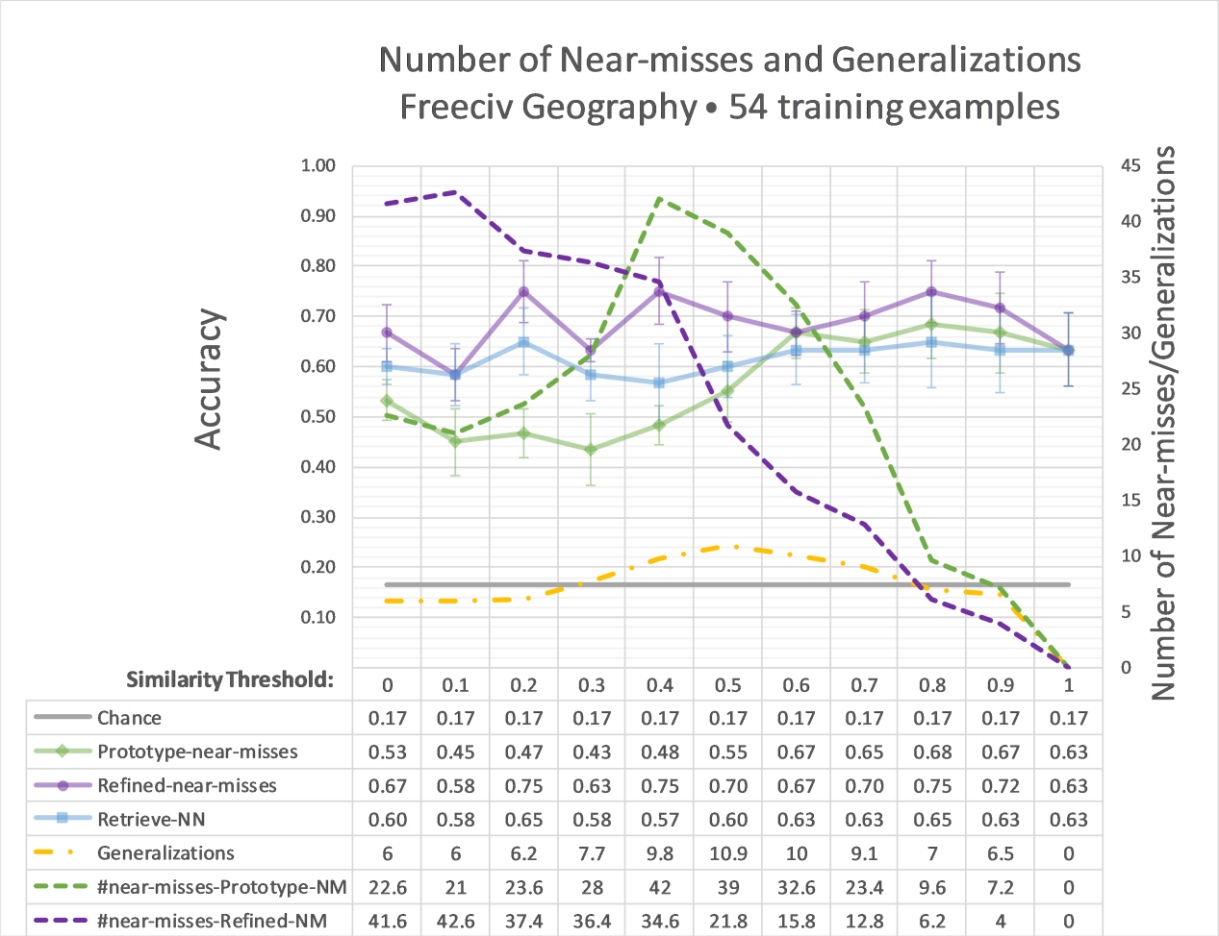
classifier had only 5 hypotheses by the end of learning. The Refined-near-misses classifier had 50.

Figure 4.9 show the accuracy, the number of hypotheses generated, and the number of valid hypotheses found by the Retrieve-NN, Refined-near-misses and Prototype-near-misses classifiers on the Freeciv Geography dataset. Here, Refined-near-misses achieved a peak accuracy of 75%, outperforming the peak accuracy of Retrieve-NN at 65% and Prototype-near-misses at 68%. These peaks occurred at the same similarity threshold (0.8) and were not



Similarity Threshold:	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Chance	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17
Prototype-near-misses	0.53	0.45	0.47	0.43	0.48	0.55	0.67	0.65	0.68	0.67	0.63
Refined-near-misses	0.67	0.58	0.75	0.63	0.75	0.70	0.67	0.70	0.75	0.72	0.63
Retrieve-NN	0.60	0.58	0.65	0.58	0.57	0.60	0.63	0.63	0.65	0.63	0.63
SurvivingHypotheses-Refined	69.9	76.5	66.1	81.3	73.8	57.9	25.8	25.6	6.4	5.7	0
SurvivingHypotheses-Prototype	140.8	132.2	125.3	117.4	135.9	142.9	72.9	23	1.4	8.1	0
AllHypotheses-Prototypes	179.9	170	157	140.2	156.1	157.5	80.8	25.4	1.4	8.1	0
AllHypotheses-Refined	81.4	84.4	73.4	87.9	79.6	61.6	27.7	26.4	6.4	5.7	0

Figure 4.9: The accuracy of the Prototype-near-misses, Refined-near-misses and Retrieve-NN classifiers, overlaying the number of near-miss hypotheses generated during learning and the number of them that were not pruned.



Similarity Threshold:	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Chance	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17
Prototype-near-misses	0.53	0.45	0.47	0.43	0.48	0.55	0.67	0.65	0.68	0.67	0.63
Refined-near-misses	0.67	0.58	0.75	0.63	0.75	0.70	0.67	0.70	0.75	0.72	0.63
Retrieve-NN	0.60	0.58	0.65	0.58	0.57	0.60	0.63	0.63	0.65	0.63	0.63
Generalizations	6	6	6.2	7.7	9.8	10.9	10	9.1	7	6.5	0
#near-misses-Prototype-NM	22.6	21	23.6	28	42	39	32.6	23.4	9.6	7.2	0
#near-misses-Refined-NM	41.6	42.6	37.4	36.4	34.6	21.8	15.8	12.8	6.2	4	0

Figure 4.10: The average number of near-miss comparisons and generalizations made during learning, overlaying accuracy, for the Freeciv dataset.

significantly different. The Refined-near-misses and Retrieve-NN classifiers showed other peaks at this same accuracy with lower similarity thresholds and were generally unharmed by dropping the similarity threshold. Though only significantly different at one similarity threshold (0.4), the Refined-near-misses approach did consistently outperform the Retrieve-NN classifier. The Prototype-near-misses classifier performed significantly worse than the others at low similarity thresholds (< 0.4). Prototype-near-misses generated roughly twice as many hypotheses as the Refined-near-misses for all thresholds below 0.7, peaking at 180 for S=0. Prototype-near-

misses pruned a greater proportion of its hypotheses than Refine-near-misses, but not enough to make up for the generation rate, resulting in Prototype-near-misses having more hypotheses that survived generalization. When generalization was occurring (S<1), the number of generalizations produced ranged from 6 to 11 (recall, there were 54 training examples), as depicted by the yellow dashed/dotted line in Figure 4.10. It was highest for middle-ranging similarity thresholds, where the rate at which generalizations are forming outpaces the rate at which they merge. The Prototype-near-misses classifier detected more near-misses than Refined-near-misses for S>0.3, and fewer for lower thresholds (the dashed green and purple lines in Figure 4.10). Whereas Refined-near-misses detects an increasing number of near-misses as the similarity threshold decreases, the Prototype-near-misses condition is limited by the number of prototypes, and thus starts detecting fewer near-misses as the similarity threshold drops below 0.4.

4.6 Discussion

On both tasks, the near-miss classifiers both achieved higher accuracy at their peaks than did a comparable similarity-based classifier, though not significantly in the Freeciv Geography task.

The relative performance of the two near-miss classifiers was reversed between the two tasks – Prototypes performed best on the first and Refined-near-misses did best on the second. However, in both tasks, everywhere where the performance of the two near-miss classifiers was significantly different, it was the one that had settled on fewer hypotheses that performed better. This suggests that too many hypothesized criteria hurt performance, even though it is also evident from the results that hypothesized criteria from near-misses can be useful in aiding sketch recognition

Notice that in Prototype-near-misses on the Freeciv Geography task, for similarity thresholds below 0.6, the number of surviving near-miss hypotheses was relatively stable. However, in this same range, the number of near-misses detected dropped with the similarity threshold (forced to by the reduced availability of cases due to compression). Clearly the extra differences in the lower-threshold near-misses were resulting in enough hypotheses to compensate for the reduced number of near-misses detected. This range is where hypothesis pruning picked up, likely due to these unreliable far-miss hypotheses.

Besides the hypotheses, the other difference between the near-miss learners is how they retrieve cases from long term memory at testing time. Refined-near-misses retrieved from all the individual training examples, while Prototype near-misses retrieved from the prototypes. Losing this granularity in the training case library could obfuscate the nearest neighbors for a testing example, per the discussion in Section 4.4.2.

However, the Retrieve-NN control condition results provide some confidence that the difference in efficacy between the near-miss learners could not be readily explained by the testing-time retrieval process, since this process is always the same between Retrieve-NN and Prototype-near-misses, and it is the same between all three for S=1. Given hypotheses, Refined-near-misses is able to outperform this control condition, and Prototype-near-misses suffers for low thresholds while the Retrieve-NN control remains consistent. In fact, the Retrieve-NN classifier is surprisingly consistent across similarity thresholds, meaning the quality of testing time retrievals is relatively consistent across all conditions tested. Thus, the quality and quantity of criteria hypotheses are left as an explanation.

One potential strategy is to only treat the top hypotheses as working hypotheses during classification. These might be chosen based on the size of the associated prototype (in terms of assimilated examples), or the nearness of the near-miss it came from.

The message here is that near-miss hypotheses are helpful and far-miss hypotheses are harmful. This is both because of the number of hypotheses generated and because as the near-miss pair gets more and more distant, the mapping between them becomes less and less reliable. So not only is the discriminative information less localized, it is noisier.

Because we tie the assimilation threshold to the near-miss threshold (together they are the similarity threshold), it is hard to say from the results here whether using analogical generalization to prune near-miss hypotheses (on the basis that they represent necessary conditions for similar examples) actually helped, or whether the pruned hypotheses would have been just as useful as the surviving ones. However, the experiment in (McLure et al. 2010) did decouple the thresholds and fixed the near-miss threshold, providing some evidence that analogical generalization is helpful for pruning. In any case, the low rate of hypothesis pruning here was surprising.

5 Support Vector Machines with Structure Mapping

The near-miss classifiers in the last chapter learned hypotheses about necessary properties of example or prototypes by comparing them to negative examples or prototypes during training. The hypothesized criteria were stored in the (positive) prototype and deployed at testing time by projecting them onto an unlabeled example to see if they held. Here, we use the linear support vector machine (SVM) with analogical generalization in an approach that learns similar hypotheses from near-miss pairs, but these hypotheses are determined entirely at testing time. Rather than rely on the positive prototype as a conduit for the criteria (from negative example to positive example), the approach here constructs an ad hoc generalized model that incorporates all three examples via analogical matching. This generalization creates a feature space where all three can be located, in order to learn and apply a separator.

The support vector machine (SVM) is a well-understood discriminative classification method for separating two classes in a feature space. The linear SVM learns a linear classification boundary – a hyperplane – in the feature space that maximizes the margin between the boundary and the nearest instances of either class. Figure 5.1 shows a two-dimensional linear SVM.

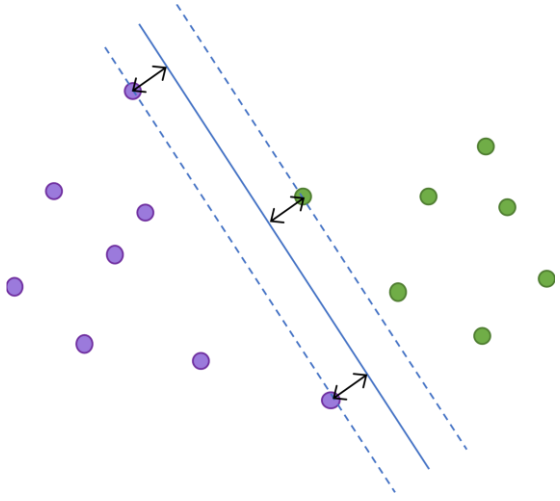


Figure 5.1: A 2-D linear support vector machine.

We are interested in the performance of SVMs built on structure mapping for several reasons. First, SVMs have a similar learning bias to the near-miss techniques in Chapter 4 (Figure 5.4

below). Second, they extend more naturally than necessary condition hypotheses to representations with uncertainty (Section 7.3.9 below). Third, the SVM learns a maximum separating hyperplane along with a set of support-vectors – a small subset of the training examples in the SVM that fully specify the hyperplane. These support vectors may be a good signal about which cases in long term memory are worth retaining as retrieval/matching candidates for near-misses and classification (a compromise between the higher performing Refined-near-misses approach and the more scalable Prototype-near-misses approach). Finally, the SVM is pursued because of its theoretical and practical foundations in the realm of active learning (Bloodgood 2018; Schohn and Cohn 2000; Tong 2001), which was motivated in Section 4 and is revisited in Section 7.3.10 below.

In this chapter we explain and test a technique called the Analogical Boundary Learner (ABLe) that uses MAC/FAC and SAGE to classify examples by training ad hoc analogy-based SVMs at testing time on a subset of labeled examples that are similar to the unlabeled example. We show that this improves on a comparable similarity-based classifier, Retrieve-KNNW (Sections 2.1.3 & 2.1.4). It outperforms Refined-Near-Misses, the best performing classifier from Chapter 4, over a wide range of similarity thresholds even though it stores fewer than half as many assertions in the knowledge base during training, and far fewer than that at lower similarity thresholds. The SVM's are built on top of LIBSVM (Chang and Lin 2011) via the CL-LIBSVM wrapper.

First, we present the general rationale behind building a feature space by analogy.

5.1 Structure Mapping through the lens of feature-vectors

Feature-vector approaches to machine learning assume an implicit one-to-one mapping between input variables across instances. When the features correspond to slots in a vector, or keys in a set of key-value pairs, or even pixels in a grid, the one-to-one correspondences between input features in one example and input features in another example are predetermined. Furthermore, these correspondences are transitive – if a feature in one example corresponds to some feature in a second example, then these two cannot correspond to two different features in a third example. This allows statistics to be tracked for the slots in the vector, or template. However, it is often not trivial to determine what slots to use for a given learning problem, and how raw input should be used to fill the slots, except in the case of pixels. It's not obvious how the pixel case would naturally extend to incorporate information beyond the pixels, such as accompanying language in a sketching context. Learning directly from pixels also tends to produce less explainable models, at least in the case of deep neural networks commonly used in computer vision – convolutional neural networks – which learn complex nonlinear functions over pixel values. Learning directly from pixels also tends to require many training examples, unless pre-trained intermediate representations are available or enough is known about a domain to perform training set expansion by modifying existing training examples.

Some learning approaches train on examples that are logical representations of any size. Here, input examples do not need to conform to some predetermined template like a feature-vector. Inductive Logic Programming (ILP) techniques (Michalski 1983; Muggleton and Feng 1992; Plotkin 1970; Quinlan 1990; Srinivasan 1999) require no such template or implicit mapping because they learn logically quantified rules (containing variables) from input examples

expressed in open-ended first-order logic. The learned concept is a set of clauses that is satisfied by the positive examples and not the negative, often incorporating background knowledge from a knowledge base. ILP reduces the need for task-specific preprocessing and yields highly explainable concepts (sets of rules), insofar as the predicates in the knowledge base are explainable. Section 6.1.2 goes into more detail about ILP. A typical challenge for ILP is that the hypothesis space to be searched is very large, as it is not constrained to be expressible by some template. Instead it is constrained by the possible combinations of constants and predicates in a knowledge base.

Structure mapping is a way to apply feature-based approaches to expressive, unconstrained logical representations. The inputs are cases, arbitrarily sized sets of open-ended predicate calculus statements. An analogical matching step automatically determines how these statements should align across two examples, base and target, with an injective mapping function, meaning correspondences must be one-to-one but there can be leftover statements (without correspondences) on either side. There is a preference to find large consistent systems of relationships. The match can produce a feature space by creating one feature for each pair of statements (one from the base, one from the target) that are found to correspond, one feature for each leftover statement in the base, and one feature for each leftover statement in the target. This set of features is a template – a structured representation where the elements (statements) in the structure are keys which can be assigned values to represent an instance. For example, the base of the match may be expressed in the template using binary values, assigning 1 to each statement in the template that has an analog in the base and 0's for all other statements in the template. However, the values need not be binary. For example, they may be the probabilities associated

with the corresponding facts in the base, or they may be some other quantity that is an argument in the assertion.

Note that this procedure for producing features mirrors how SAGE generalizes two cases based on a match between them. Indeed, we use analogical generalization (implemented in SAGE) to form the template and extract feature vectors. Additional examples can be assimilated into the template by further analogical generalization, incrementally expanding the feature space to accommodate the leftover elements in each new example and back-filling values for those new elements in the feature vectors of previously assimilated instances. This means that every element that shows up in an instance that is assimilated into the template must have a corresponding element in the template. This provides an elegant answer to the question of how to deal with hypotheses about something hypothetical – a leftover entity from the negative side of a near-miss that has no analog in the positive side. The near-miss learners from the last chapter, which embedded hypotheses about a concept in its positive examples, dealt with this by introducing constrained variables (Section 4.2.1), much like ILP. Using analogical generalization to make a template that includes positive and negative examples means there will not be anything extra in any of the assimilated examples, because the template grows to accommodate them upon assimilation.

Statistics can be tracked for elements in this template, just like with slots in a feature-vector. This was Halstead's fundamental insight (Halstead 2011), and the basis of Liang and Forbus's technique for structured logistic regression, SlogAn (Liang and Forbus 2015). We compare further to the analogical learning approaches of Liang & Forbus (2015) and Halstead (2011) in Section 6.1.1.

5.2 The Analogical Boundary Learner

Tractability is a concern when SVM's are trained on many instances. The SVM-KNN technique (Zhang et al. 2006) addresses this by, given an example to classify, first finding the k-nearest neighbors using a similarity measure and subsequently training local SVM's among the k training examples. We use this same strategy in ABLe, where the SVM tractability concern is eclipsed by the costly process for assigning feature vectors to instances. Specifically, assigning an instance a vector requires matching the instance to the template via an SME comparison and merging the instance into the template based on the best mapping in the match. MAC/FAC is used for the similarity-based retrieval of k neighbors.

The training phase in this approach uses the same SAGE-based paradigm used by the Retrieve-KNNW approach in Section 2.1.4. In this paradigm, a SAGE generalization pool is maintained for every label, all of which share an assimilation threshold, and training examples are added one by one to the pools corresponding to their labels, ultimately clustering and compress them into a set of prototypes per label.

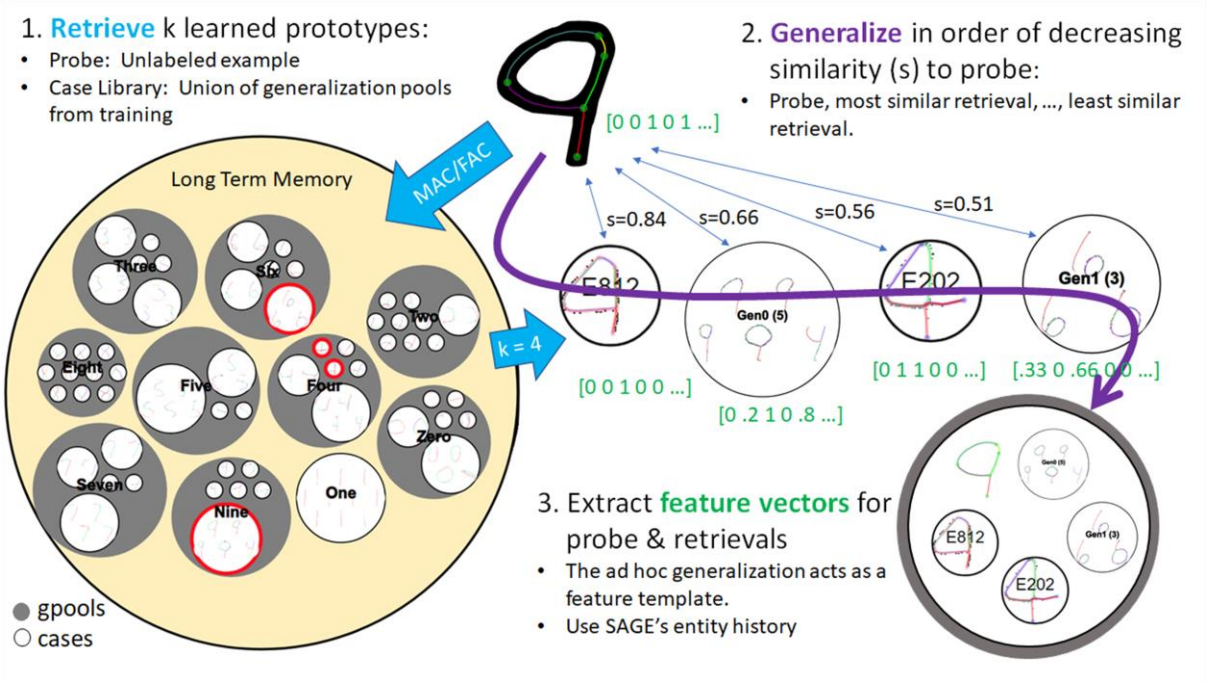


Figure 5.2: The three steps in ABLe for setting up a support vector machine to classify an unlabeled example.

In the testing phase, given an unlabeled example, an ad hoc SVM is constructed to classify the example based on k prototypes from training that are similar to it. The steps for constructing this SVM are outlined in Figure 5.2. First, k similar cases (examples or generalizations) are retrieved from the training case library using MAC/FAC with the unlabeled example, just like in the Retrieve-KNNW, Prototype-near-misses, and Refined-near-misses approaches.

Second, the k+1 cases are sequentially added to an ad hoc SAGE generalization pool with an assimilation threshold of zero, which will produce a single generalization. Below we will call this the *template generalization*. The unlabeled example is added first in order to seed the template generalization (giving it an impact on how all the k retrieved examples are generalized). In order to further anchor the generalization on the unlabeled example, the rest of the examples are added to the pool (and always generalized because of the zero threshold) in decreasing order

of similarity to the unlabeled example, using the similarity scores already embedded in the MAC/FAC results.

Third, a feature space is created from the template generalization. Each assertion in the generalization becomes a feature (a dimension in the space). Each of the k+1 cases that were assimilated into the generalization is assigned a set of values for these features based on whether it contains an analog for the assertion. Specifically, to fill the slot in a case's feature vector that corresponds to a given template assertion, we use SAGE's entity history (Section 4.4.1) to translate the assertion to the case, i.e. replace generalized entities in the template assertion with their corresponding entities in the case. If the translated assertion does not hold in the case, the slot is assigned a zero. If the translated assertion holds in the case and has a probability associated with it (as in a generalization from training), the slot is filled with the probability. Otherwise, the slot is filled with a one.

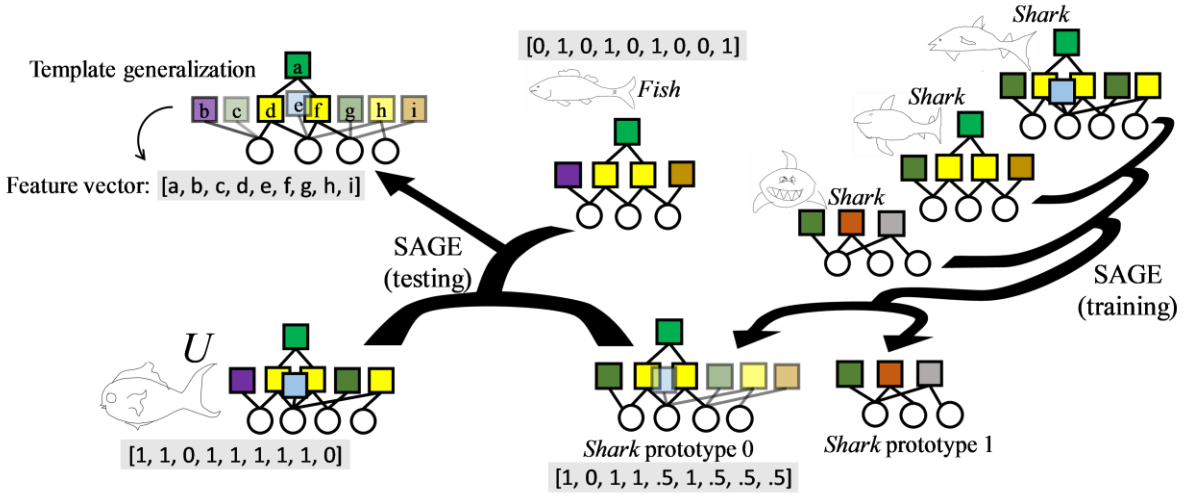


Figure 5.3: Two separate roles of SAGE in ABLe: one in training, one in testing.

To be clear, note that ABLe uses SAGE in two separate roles (Figure 5.3). SAGE in training clusters examples into prototypes and SAGE in testing builds a template generalization from a subset of those prototypes.

Here we use LIBSVM's (Chang and Lin 2011) built-in *n*-ary classification method of an ensemble of *n(n-1)* one-vs-one classifiers that make weighted votes on the *n* labels. It is straightforward to implement one-vs-all *n*-ary classification with LIBSVM, but here we chose one-vs-one for convenience, and for the same reason it was chosen by the LIBSVM authors – it has a faster training time and ostensibly comparable performance. By using one-vs-one we intended to able to run more trials to get a fuller view of learning curves, variance, and the effect of generalization, and end up with a proof of concept of running an SVM on a SAGE generalization. As it turned out, we had a different bottleneck that kept k too low for this to matter much in these experiments. However, that bottleneck can be addressed with engineering (Section 7.3.3), so the tractability of training the SVM with respect to k remains relevant.

Nonetheless, the one-vs-all approach is an interesting avenue of future work, justified in Section 7.3.7 below.

We used ridge (L2) regularization, though as we will point out in Section 7.3.2, lasso (L1) regression may be a more interesting avenue going forward due to its simpler, more explainable hypotheses.

If the set of k examples that make it into the SVM contain a highly similar, differently labeled pair, the maximum separating hyperplane acts a lot like a criteria hypothesis born of a near-miss comparison between the pair in the classifiers from the last chapter. The SVM, like the criteria hypotheses, will hypothesize a separator such that the property(ies) separating the near-miss pair will remain critical in other locales of the feature space. Figure 5.4 shows three labeled training examples and an unlabeled example, plotted on a three-dimensional cube. The

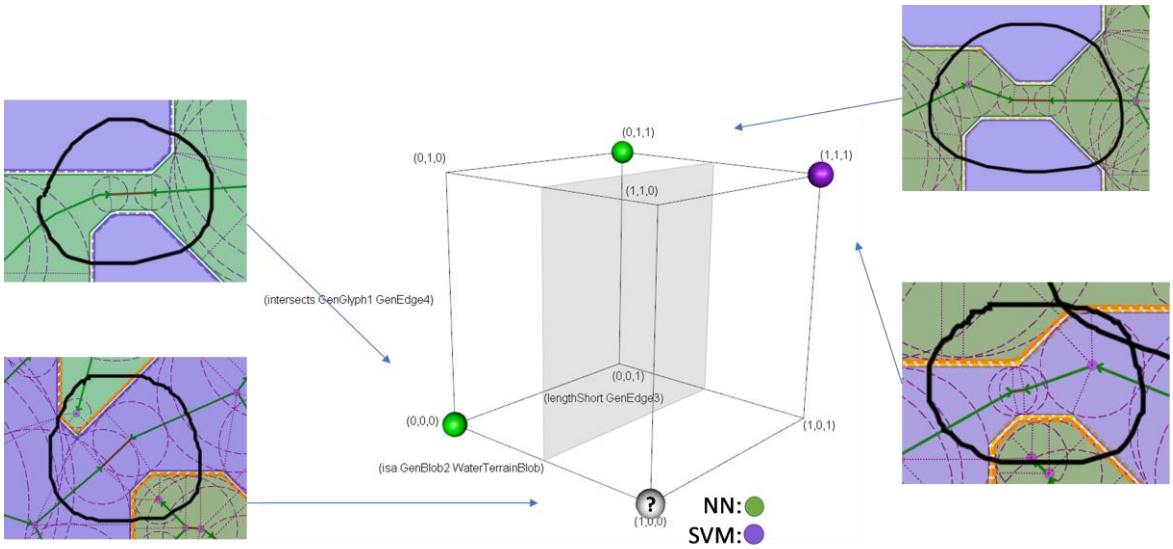


Figure 5.4: Two isthmuses (top) and two straits (bottom), and their locations in (a 3-D projection of) a feature space that is an n-dimensional unit hypercube, where n is the number of assertions in the template generalization. These examples are assumed to agree on every dimension (assertion) not shown. A linear SVM on the hypercube would disagree with a nearest-neigbor classifier in this case.

dimensions shown are assigned to assertions in the template generalization as follows: The x-axis is `(isa GenBlob2 WaterTerrainBlob)`, the y-axis is `(intersects GenGlyph1 GenEdge4)`, and the z-axis is `(lengthShort GenEdge3)`. (The hypothetical encoding scheme used on these four examples has described them as the same on every dimension in the n-dimensional hypercube that is not shown, so we can ignore them.) The maximum margin hyperplane, given the three training examples, has been entirely determined by two of the examples that differ only on the x-axis and have different labels, *isthmus* and *strait* (a near-miss pair). The unlabeled example is most similar to the third training example (an isthmus) but differs from it along this same critical dimension (the x-axis) and is therefore separated from it by the hyperplane. A nearest-neighbor classifier would call it an isthmus, whereas ABLe calls it a strait.

5.2.1.1 A linear kernel

ABLe uses linear kernels in its SVMs. Figure 5.5 shows some hypotheses that a linear SVM can learn in the feature space created by assertions A, B and C in a generalization template (again, we assume that the training cases are identical on every other dimension besides A, B and C). These hypotheses are maximum-margin hyperplanes that cut through the unit hypercube where there is a dimension for every assertion in the template generalization. The cases in the training case library (in the figure, represented by colored spheres) fall at the vertices of the

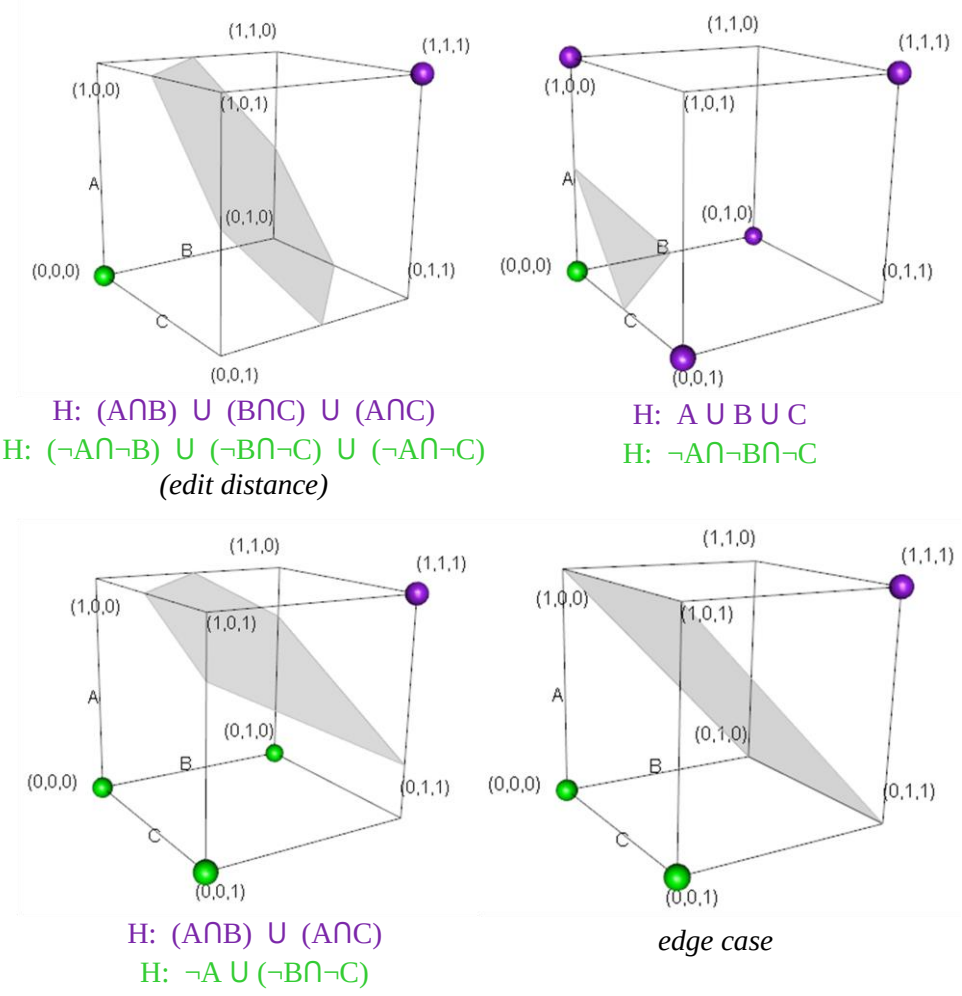


Figure 5.5: Hypotheses representable in a support vector machine where dimensions are assertions, trained on examples from two categories (green and purple).

hypercube unless they have continuous values associated with their assertions, like in a SAGE generalization, in which case they can fall anywhere inside the unit hypercube. The testing examples in the present experiments never have continuous values, so a hypothesis separating two labels can be expressed as a pair of logic sentences, one per label. The logic sentences are necessary and sufficient conditions for membership in a binary choice between labels, based on how the hyperplane separates the vertices of the hypercube. Note that in a multiple-category task, the exact location of the hyperplane still matters (beyond how it separates the vertices) because the votes from the one-vs-one classifiers in the ensemble are weighted by the distance from the unlabeled example to the hyperplane. A notable difference between these hypotheses and the criteria hypotheses from the near-miss classifiers in the last chapter is that the logic sentences can be disjunctive, as opposed to relying on similarity to organize a disjunctive definition.

The hyperplanes in the figure are for the hard-margin SVM, but ABLe uses a soft-margin SVM, where the margin width trades off with a penalty for training cases that violate the margin. In effect, this allows more distant examples – those that are not support vectors – to still have a minor influence on the location of the hyperplane. For example, in the hypercube in the bottom right, this would allow the farther green case to break the tie at the four ambiguous vertices by tilting the hyperplane slightly. The C parameter in a soft-margin SVM – ranging 0 to infinity – controls the balance between the margin size and minimizing violations. As C approaches infinity, the soft-margin SVM becomes the hard-margin SVM. In these experiments, C=1.

In creating a feature for every assertion in a generalization, we have a very high dimensional feature space. We also use a very low k in our proofs of concept. Thus, the data is extremely likely to be separable with a hyperplane, even a linear one. Overfitting is a concern.

The XOR problem with linear classifiers is very relevant for learning concepts from qualitative spatial features, because an unencoded or filtered latent feature may be the critical one. Latent spatial features can often be expressed in other features as an XOR. For example, a *same length* relationship can be expressed in terms of a mutually exclusive and collectively exhaustive pair of length attributes *short* and *long*: sameLength(x, y) <= XOR(long(x), short(y)). The encoding schemes here make such a concept learnable by encoding *sameness* as a higher order relation, e.g. (approximately (LengthFn <x>) (LengthFn <y>)), for attributes known in the ontology to capture qualitative degrees, like length attributes. Some primitive spatial relationships, such as parallel, already capture sameness of some attributes. However, there are more interpretations of sameness beyond attributes, and likely many latent unencoded relationships that are subtler than sameness.

Feature construction may be a promising route if a potential XOR (true) concept could be detected. This might involve detecting violated concept boundaries with characteristic support vectors and using active learning via query synthesis to test them. However, more often in the current approach, noise would be an obstacle here because in such high-dimensional space, spurious differences can explain away an XOR relationship that would cause a lot of margin violation by causing an orthogonal (or nearly orthogonal) hyperplane to be found. One question then is how to gather candidate sets of properties for constructing new features from their (higher-order) relationships, and which relationships to test. We leave such active learning opportunities for future work, but nonetheless use their promise as justification for a linear kernel, which is more amenable to communicating learned concepts, serving the goal of explainability (the normal of hyperplane in a linear SVM can be expressed in terms of weights

on dimensions, which are assertions embedded in a structured model – a simple way to understand which are the most critical properties, especially if the model can be visualized).

5.2.1.2 Anchoring on the unlabeled example

One interesting aspect of SAGE is that the order in which the examples are assimilated affects how the template develops, because each example is assimilated into the template by matching against the template itself. In this way, the comparison that assimilates each example into the template is impacted by every example that was assimilated before it. The first two examples to be generalized can be thought of as anchoring the template.

One assumption in ABLe is that it is best to anchor the template generalization on the unlabeled example (U). The point in the feature space corresponding to U is the location that we are ultimately trying to judge. The bet here is that making U as influential as possible in organizing the feature space will produce a more effective feature space for labeling U.

Furthermore, U may be a useful analogical bridge for synthesizing information from structurally dissimilar prototypes. Consider what happens as the training-phase assimilation threshold is lowered. More clustering occurs, and consequently, the cases in a generalization pool become fewer and often farther between. For some concepts – especially complex concepts (cars as opposed to nines) – we expect that at a certain point in dropping the threshold, the cases will become significantly less productive since they will have been constructed by shoe-horning un-analogous cases into analogies.

Now, consider what we are doing with SAGE in the testing phase. We retrieve cases (examples or generalizations) similar to it using repeated calls to MAC/FAC, then force a generalization across all the results. Any retrieved cases that come from the same category (let's

say cases C_a and C_b) must have been separate clusters in the same generalization pool. We can reasonably expect them to look quite different, which raises the same shoe-horning concern. On the other hand, they were each similar enough to the unlabeled example U to get retrieved. C_a and C_b may each share common structure with U, just different parts of U. By beginning the generalization with U, we guarantee that these two cases will have the opportunity to map onto independent subsets of the elements in U. Each case will also have the opportunity to map onto structural elements that came from any of the examples that were assimilated before them. As it gets to less and less similar cases, the template representation gradually expands to more flexibly assimilate them.

One reason we concern ourselves with how to assimilate structurally dissimilar examples is that we currently use all k of the cases retrieved by MAC/FAC to build the SVM. This means the content vector dot product (MAC) entirely determines which examples are used in the SVM. This is vulnerable to false positives (structurally dissimilar examples that make it into the generalization template One question for future work is whether an additional structural-similarity based filter is helpful (e.g. from the k retrievals, select the J with the highest similarity scores). Here, we use the simpler approach and ignore this extra parameter. We return to a related idea in Section 5.5.1.

5.3 Experiment 4: Comparison to Similarity-based classification

This experiment used a sketch recognition task on three datasets to compare ABLe to a Retrieve-KNNW classifier (Sections 2.1.3 and 2.1.4), which gathers the same number of retrievals as ABLe, and sums up the similarity scores of the k retrievals to determine a label instead of training an SVM. The k in all three experiments was 9.

The first task was to classify the Berlin25 dataset (Section 2.3.1) over a range of assimilation thresholds (0.4 to 1). Computation of the template generalization was too intractable to gather results for assimilation thresholds under 0.4. This dataset was used with the top performing encoding scheme from Experiment 1 – a hybrid edge/edge-cycle encoding with an ink coverage filter of size 100. A 4-fold cross-validation structure was used, where the examples for each concept were spread evenly across the folds. Chance was 4%.

The second task was classifying the Freeciv Geography dataset (Section 2.3.2) over a range of assimilation thresholds (0 to 1). A 10-fold cross-validation structure was used, where the examples for each concept were spread evenly across the folds. Chance was 17%.

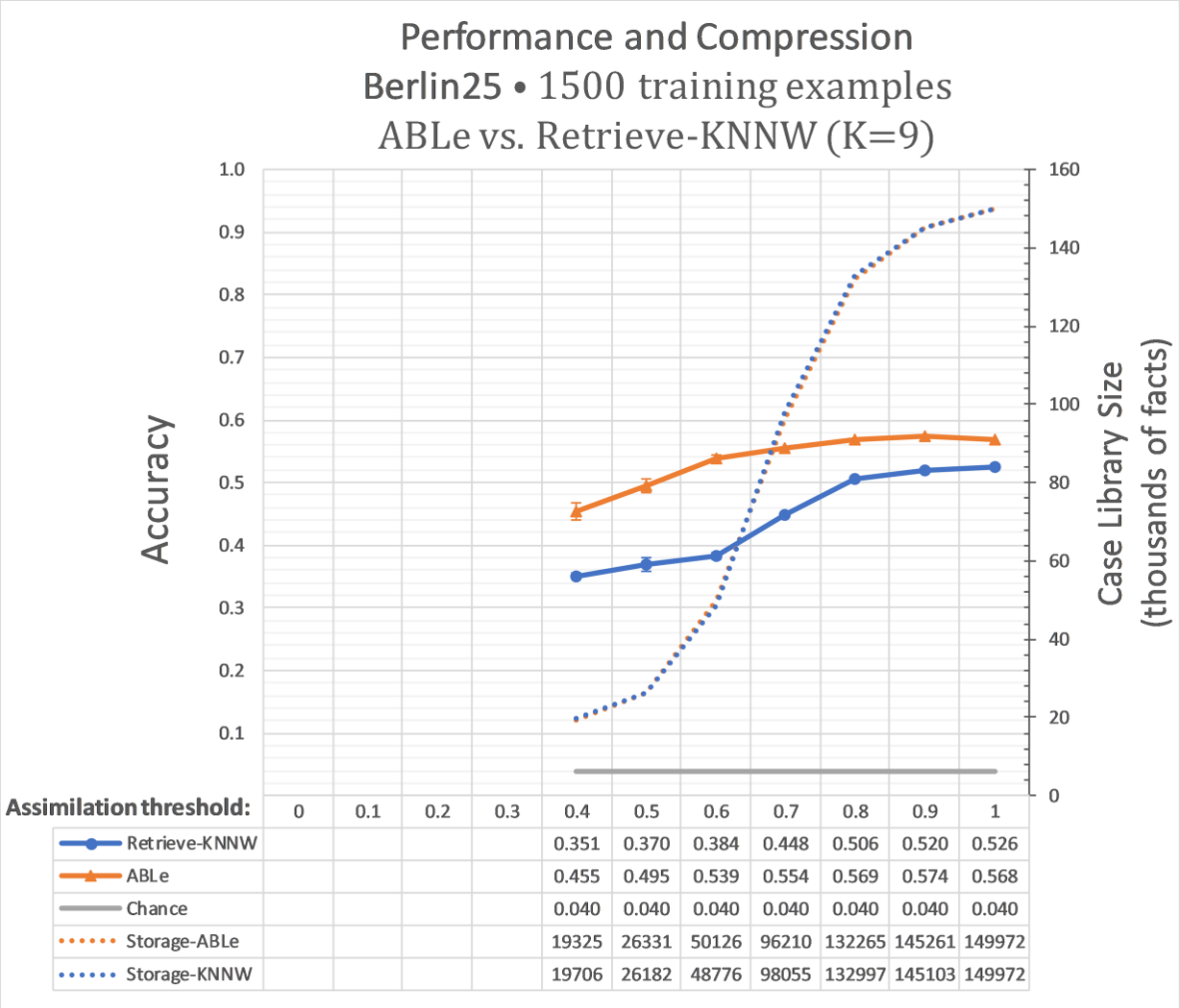
The third was classifying the MNIST dataset (subset) at two assimilation thresholds, 0.5 and 1. Here, four trials were run for each condition, in which the training set was selected at random from the (reduced) MNIST training set, and the testing set, consisting of 50 examples of each concept (500 total), was randomly selected from the (reduced) testing set. The MNIST examples (after the skeleton extraction process in Section 2.3.5) were encoded using the Hybrid scheme from Section 3.2 and filtered to a limit of 100 facts using the ink coverage filter from Section 3.3.2. Chance was 10%.

In each task, we tracked the total storage requirement – the number of assertions in the training case library (the union of the generalization pools for all categories). This provided a measure of compression over the range of assimilation thresholds. The training process is the same for ABLe and Retrieve-KNNW, so the number of assertions in the training case library is approximately equal.

For the MNIST dataset, we also tracked the average time it took to classify an example.

5.3.1 Results

On all datasets, for all assimilation thresholds (S), classification by training a SAGE-based SVM on the k retrievals (ABLe) outperformed classification using a weighted sum of their structural similarity scores (Retrieve-KNNW). Both classifiers tended to perform worse for lower assimilation thresholds, but on the Freeciv Geography dataset, the Retrieve-KNNW

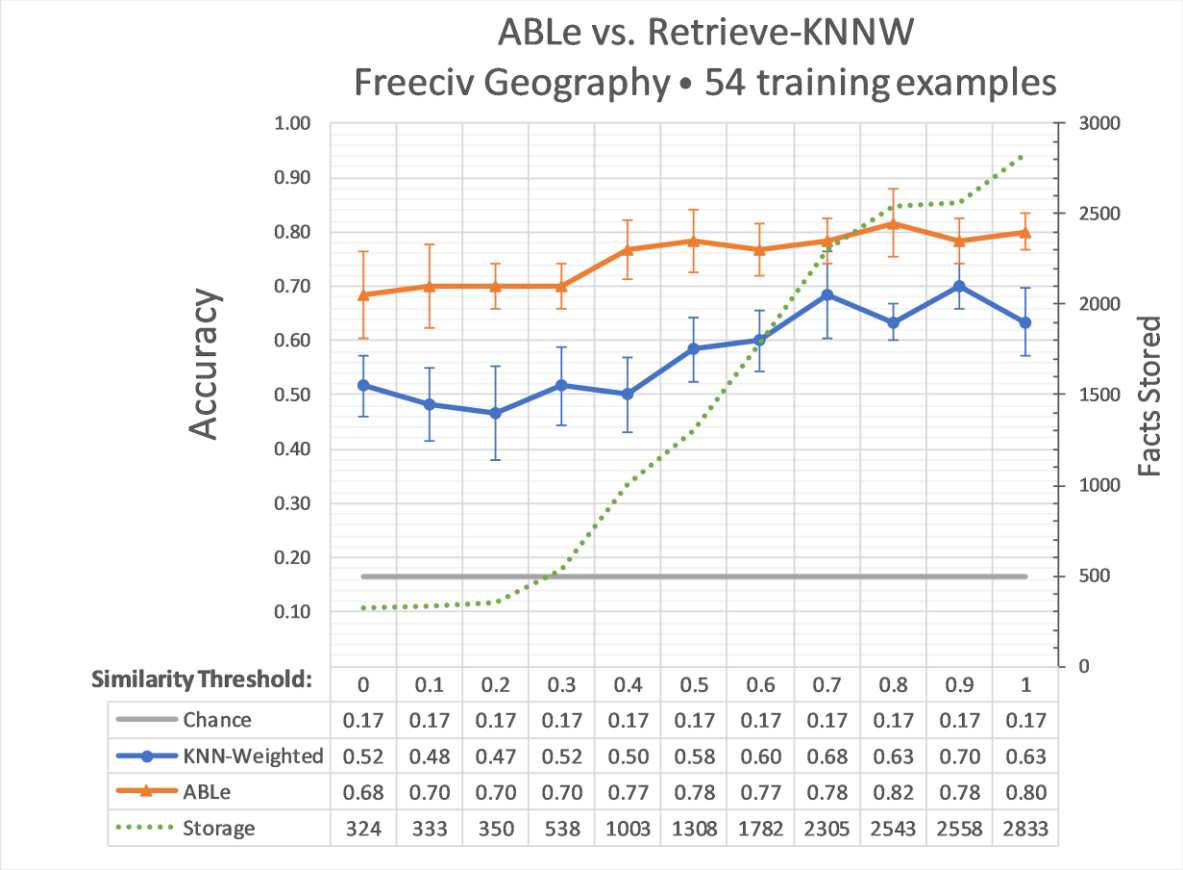


Assimilation threshold:	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Retrieve-KNNW					0.351	0.370	0.384	0.448	0.506	0.520	0.526
ABLe					0.455	0.495	0.539	0.554	0.569	0.574	0.568
Chance					0.040	0.040	0.040	0.040	0.040	0.040	0.040
Storage-ABLe					19325	26331	50126	96210	132265	145261	149972
Storage-KNNW					19706	26182	48776	98055	132997	145103	149972

Figure 5.6: ABLE and Retrieve-KNNW classification performance on the Berlin25 dataset, over a range of assimilation thresholds (lower thresholds mean more compression by SAGE during training). The dotted curves and right axis show the compression in terms of the number of facts in the training case library. The compression is virtually indistinguishable between the classifiers because their training procedure is the same.

classifier performed worst for assimilation thresholds 0.3 to 0.5 and recovered somewhat by the time the threshold dropped below S=0.3, and the storage leveled off.

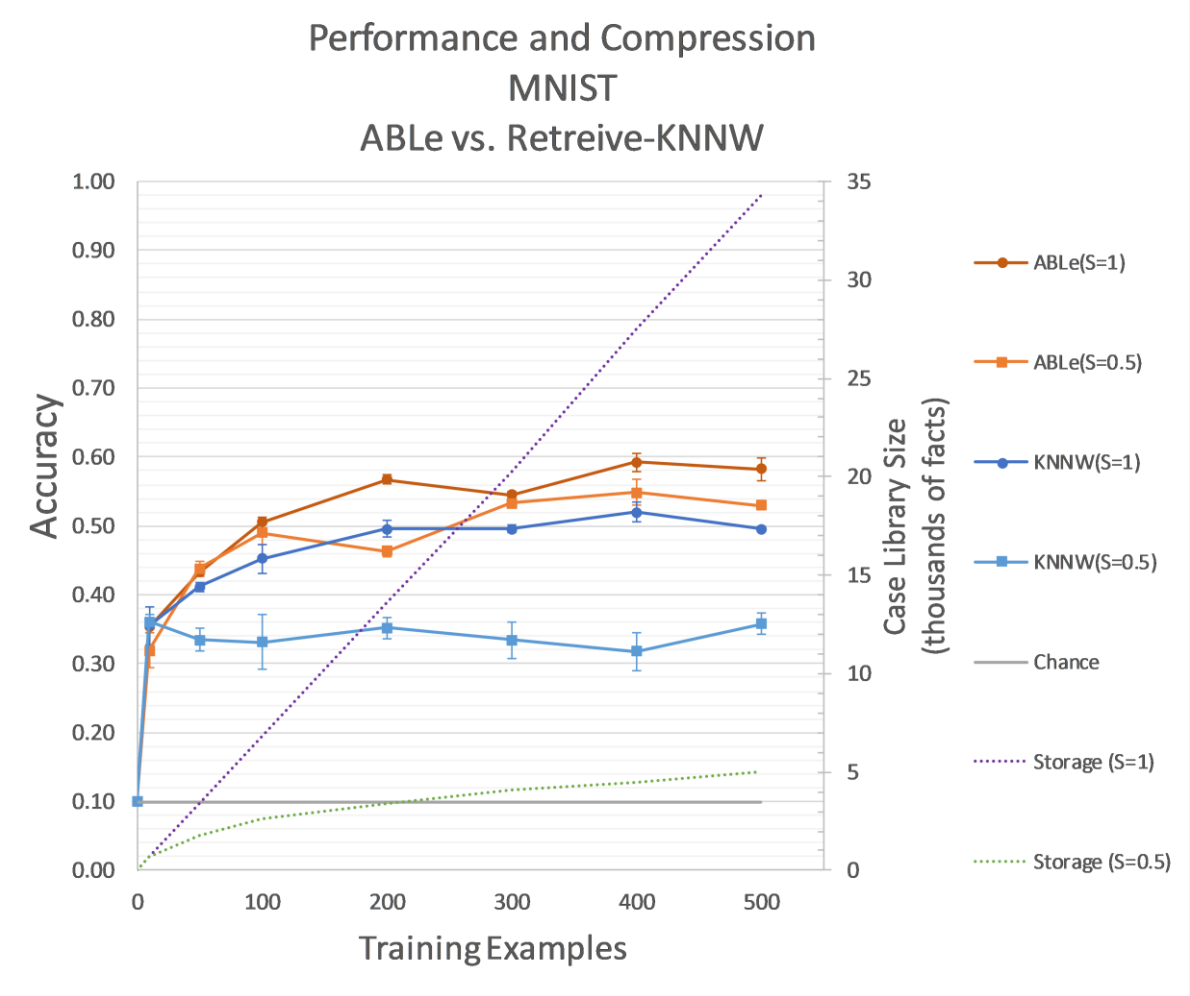
ABLe especially outperformed on the middle range of assimilation thresholds. As the assimilation threshold was lowered, when the storage requirement reached about half of its maximum and was dropping most precipitously (~0.65 on Berlin25 and ~0.55 on Freeciv Geography), the performance of Retrieve-KNNW dropped precipitously with it, whereas ABLe



Similarity Threshold:	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Chance	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17
KNN-Weighted	0.52	0.48	0.47	0.52	0.50	0.58	0.60	0.68	0.63	0.70	0.63
ABLe	0.68	0.70	0.70	0.70	0.77	0.78	0.77	0.78	0.82	0.78	0.80
Storage	324	333	350	538	1003	1308	1782	2305	2543	2558	2833

Figure 5.7: ABLE and Retrieve-KNNW classification performance on the Freeciv Geography dataset, over a range of assimilation thresholds (lower thresholds mean more compression by SAGE during training). The green curve and right axis show the compression in terms of the number of facts in the training case library (for both conditions since they share the same training process).

remained relatively resistant to the impact of compression. For example, on Berlin25 (Figure 5.6), from S=1 to S=0.6, the storage dropped by 0.67%. Over that same window, Retrieve-KNNW lost 29% of its peak improvement over chance, and ABLe lost 5% of its improvement



Training examples	10	50	100	200	300	400	500
ABLe(S=0.5)	0.32	0.44	0.49	0.46	0.53	0.55	0.53
ABLe(S=1)	0.35	0.43	0.51	0.57	0.55	0.59	0.58
KNNW(S=0.5)	0.36	0.33	0.33	0.35	0.33	0.32	0.36
KNNW(S=1)	0.36	0.41	0.45	0.50	0.50	0.52	0.50
Storage (S=0.5)	705	1751	2640	3408	4124	4440	4989
Storage (S=1)	702	3360	6785	13616	20227	27441	34283

Figure 5.8: ABLe and Retrieve-KNNW classification performance on the MNIST dataset, for two assimilation thresholds, 0.5 and 1.0. The green curve and purple curves, on the right-side vertical axis, show the compression in terms of the number of facts in the training case library for S=0.5 and S=1, respectively.

over chance. By S=0.4, storage had dropped by 87% from its peak, accuracy of Retrieve-KNNW had dropped by 36%, and ABLe's performance had dropped by 21%, closing the gap.

A similar pattern was evident in the Freeciv Geography results (Figure 5.7). As the assimilation threshold was lowered, Retrieve-KNNW dropped below its peak performance to a marginally significant degree by S=0.7, and substantially by S=0.5. ABLe didn't drop significantly below its peak performance until S=0.3.

Although we did not measure over a wide range of similarity thresholds on the MNIST dataset, we still saw evidence that ABLe was more resistant to the negative impact of compression (Figure 5.8). At an assimilation threshold of S=1, after learning on 500 examples, ABLe achieved 58% accuracy and Retrieve-KNNW achieved 50%. At S=0.5, ABLe's accuracy dropped to 53%, giving up 10% of its improvement over chance, whereas that of Retrieve-

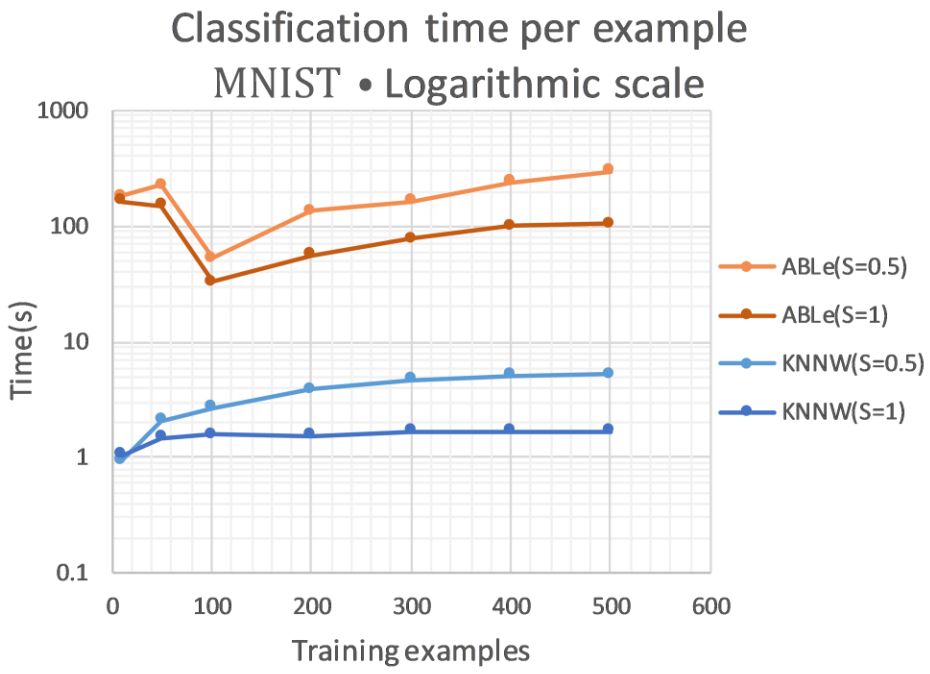


Figure 5.9: Average real time taken to classify an example in MNIST.

KNNW dropped to 36%, giving up 35% of its improvement over chance. More compellingly, one can tell from the plotted results that the Retrieve-KNNW classifier with S=0.5 didn't seem to do any better after the first 10 training examples – it stopped learning. Without compression, Retrieve-KNNW kept learning, albeit with quickly diminishing returns. ABLe at S=1 (without compression) learned best (significantly) after 100 training examples, but even at S=0.5, ABLe learned better than Retrieve-KNNW without compression. It turned out that at S=0.5, the compression rate was quite aggressive – by 500 training examples the compression ratio was 1/7. With this level of aggressive generalization, the similarity-based classifiers inability to learn is perhaps less surprising than ABLe's ability to learn on par with the uncompressed classifiers.

The average time taken to classify an example in the MNIST dataset (Figure 5.9) showed that ABLe took anywhere from 1 to 2.5 orders of magnitude longer to perform classification than its similarity-based counterpart, Retrieve-KNNW (minutes as opposed to seconds). At first glance this seems prohibitive for real-world applications, but a closer look revealed that the lion's share (84-98%) of the difference was explained by SAGE's merge step, which is theoretically fast (linear to case size as opposed to SME which is $O(n^2\log(n))$) but inordinately slow in our implementation for surmountable engineering reasons that we address in Section 7.3.4.

5.4 Experiment 5: Comparison to near-miss-based classification.

This experiment compared ABLe to the Refined-near-misses and Prototype-near-misses classifiers. We ran ABLe on the same two recognition tasks from Experiment 3 – the Sketched Concepts 2010 dataset and the Freeciv Geography dataset – in order to get an apples-to-apples comparison to the near-miss-based classifiers Refined-near-misses and Prototype-near-misses.

As in Experiment 3, the 2010 dataset the cross-validation was 4-fold and the Freeciv cross-validation was 10-fold. Chance was at 21% and 17%, respectively.

ABLe was used with k=9 (for each testing example, nine similar cases were retrieved to build and train the SVM). This was considered the most comparable setting because the near-miss learners call MAC/FAC up to 3 times for each testing example, retrieving up to 9 cases total.

We tracked the total amount of facts stored for each condition, which included near miss hypotheses for the near-miss classifiers and case libraries for all three (generalization pools for Prototype-near-misses and ABLe, and generalization pools + training examples for Refined-near-misses).

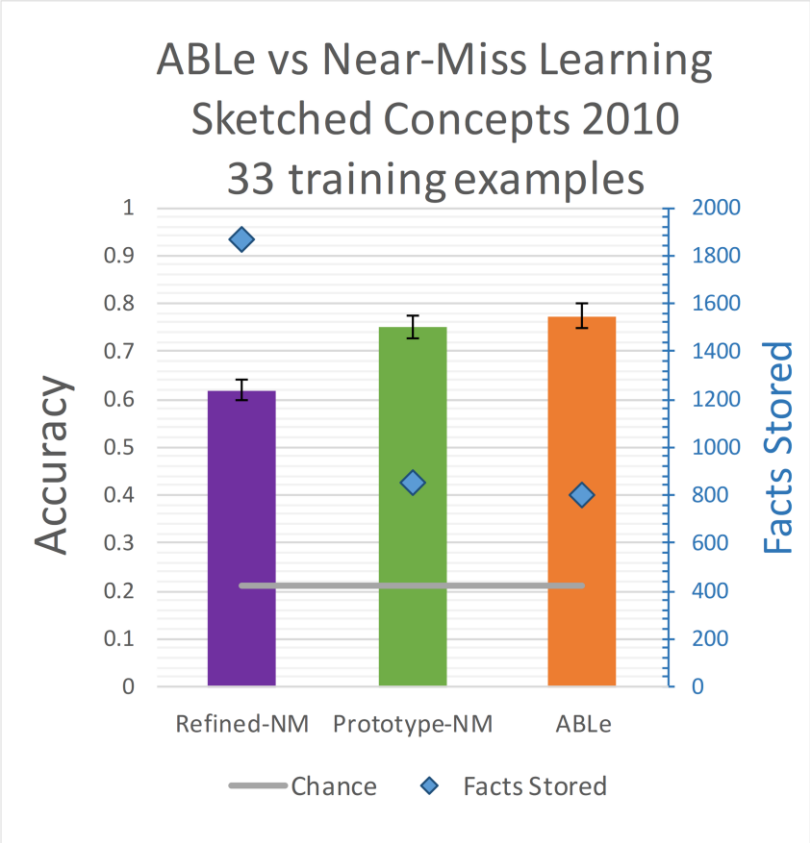
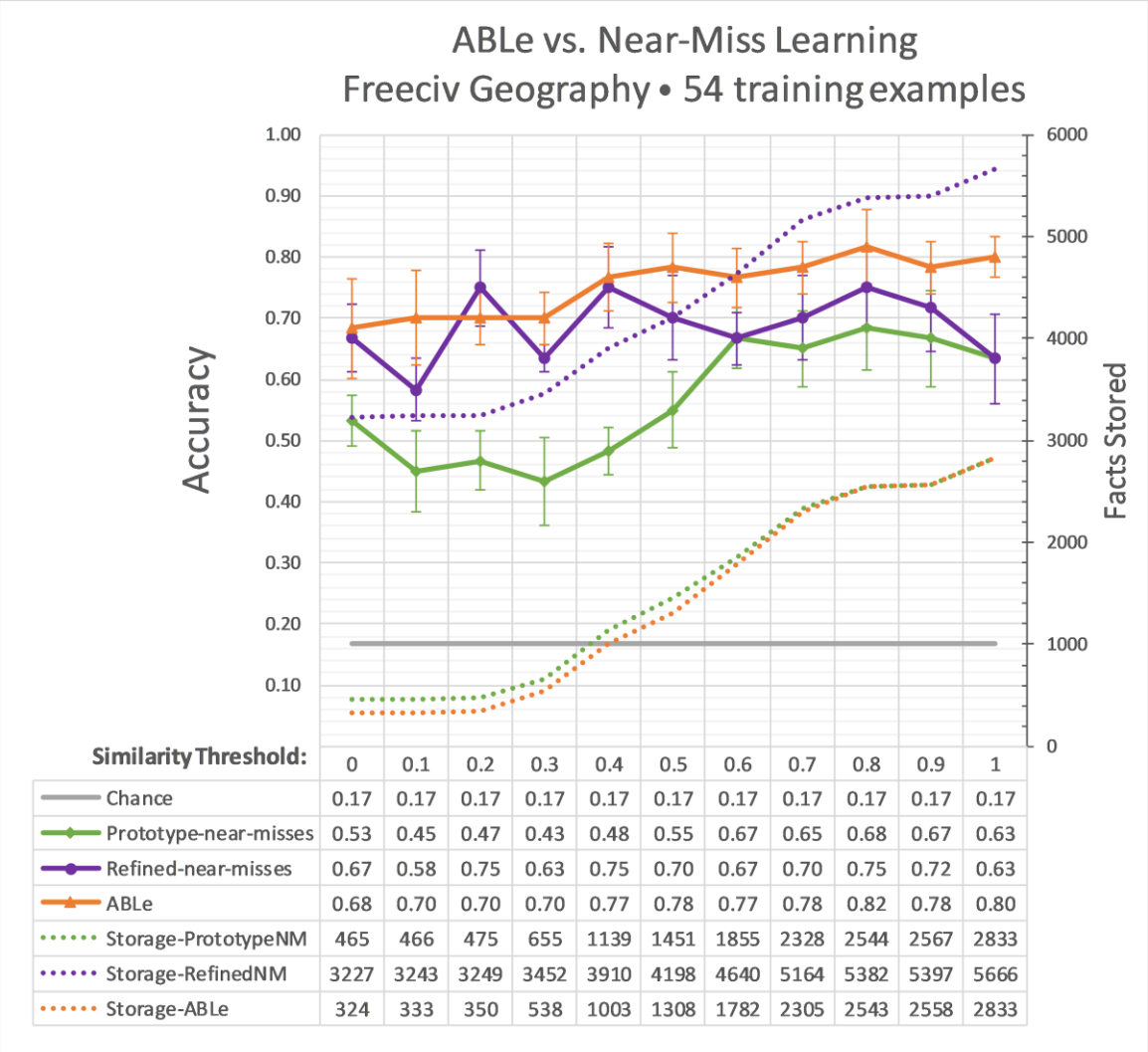


Figure 5.10: Results of ABLe on the Sketched Concepts 2010 dataset, stacked up against the near-miss classifiers from Experiment 3.

5.4.1 Results

On the Sketched Concepts 2010 dataset (Figure 5.10), ABLe outperformed both Refined-near-misses and Prototype-near-misses, but the difference with the latter was insignificant. On Freeciv Geography (***Figure 5.11***), ABLe outperformed both near-miss based classifiers for nearly all similarity thresholds, albeit an insignificant difference from Refined-near-misses for any single similarity threshold in isolation except for S=1. Even in ranges where Refined-near-misses was on par with ABLe, the latter's performance was more stable as the similarity threshold changed.

On both tasks, ABLe achieved classification accuracy that was marginally better than the more accurate near-miss classifier while using less storage than the more space-efficient near-miss classifier (the same amount as Retrieve-NN and Retrieve-KNNW).



Similarity Threshold:	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Chance	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17
Prototype-near-misses	0.53	0.45	0.47	0.43	0.48	0.55	0.67	0.65	0.68	0.67	0.63
Refined-near-misses	0.67	0.58	0.75	0.63	0.75	0.70	0.67	0.70	0.75	0.72	0.63
ABLe	0.68	0.70	0.70	0.70	0.77	0.78	0.77	0.78	0.82	0.78	0.80
Storage-PrototypeNM	465	466	475	655	1139	1451	1855	2328	2544	2567	2833
Storage-RefinedNM	3227	3243	3249	3452	3910	4198	4640	5164	5382	5397	5666
Storage-ABLe	324	333	350	538	1003	1308	1782	2305	2543	2558	2833

Figure 5.11: The performance of ABLe on the Freeciv Geography dataset, compared to that of the near-miss classifiers from Experiment 3. The dotted lines show the facts stored during training for the classifier of the corresponding color. The extra facts that separate Prototype-near-misses from ABLe for low similarity thresholds are the near-miss hypotheses.

5.5 Discussion

ABLe was more accurate than a comparable similarity-based classifier and a near-miss-based classifier that used a comparable amount of space. It proved to be more resistant to compression in the training case library than both of these alternatives. In fact, ABLe (marginally) outperforms the higher performing near-miss classifier while using marginally less storage than the near-miss classifier with the smaller footprint, combining the benefits of both. ABLe is also more stable for a changing similarity threshold, which may make this open parameter more easily tunable.

The results were proof of concept that an SVM built on analogical generalization can leverage discriminative structural properties to improve on a similarity-based approach. The linear SVM has particularly useful outputs – the normal to the hyperplane provides a vector of weights quantifying the discriminative value of each property (potentially useful for explainability), and the support vectors are the training examples that fully determine the hyperplane (potentially useful for explainability or selectively retaining valuable assimilated examples).

A review of the MNIST confusion matrices revealed that in that task, ABLe found zeros very hard to recognize and easy to confuse. It turned out that many zeros in the dataset ran up against a systematic failure in the medial-axis transform computation described in Section 2.2.2. In a circular region with a circular shaped hole (a donut), the wave-fronts of the grassfires collide all at once along a circle. Our medial axis transform is an approximation because of its baked-in pruning strategy, but this special case caused it significant problems. Using an exact medial axis

algorithm with subsequent pruning would probably be superior but detecting such special cases before computing the grassfire could be effective as well.

ABLe provides space savings in long term memory during training but pays a price at testing time when many comparisons are required in order to retrieve a group of training examples and assimilate them into the generalization template. Even though in practice the engineering bottleneck in the testing phase was the merge step, due to knowledge base transactions (addressed again in Section 7.3.4), the more relevant theoretical bottleneck is the number of matches which must take place. There are k matches during the (KNN) retrieval process, whose similarity scores are used to put them in order for assimilation into the template. Only one of these matches can be reused during assimilation – the most similar one. Therefore, ABLe required 2K-1 comparisons to classify an example, the second half of which were increasingly expensive because the template was growing. Section 7.3.1 describes a particularly exciting direction for unleashing a higher k value while performing arbitrarily few comparisons, at the cost of introducing noise and very minor extra storage requirements.

5.5.1 Pitfalls of a Rigid k Value

In Experiment 5, k=9 may have been excessive for such small training sets (33 examples in the 2010 dataset and 54 in Freeciv) spread over so many categories (7 categories including the *null* label, and 6 categories, respectively). On the 2010 dataset, some categories had as few as three training examples.

The use of a rigid number of retrievals to go into the generalization template is a potential issue for middle-ranging assimilation thresholds, because if there are few categories and many retrievals that have been clustered into prototypes by similarity, we are likely forcing differently

structured prototypes from the same category to assimilate into the same (template) generalization. This could be innocuous, especially if the disparate prototypes from the same category share separate substructures with the unlabeled example. However, it is likely to be a problem if the only reason the second prototype is being forced into the generalization (despite being substantially less similar to the unlabeled example) is because the requisite number of retrievals must be met. This is bound to weaken the reliability of the SVM in cases with relatively few categories. One potential fix is to not use a SAGE assimilation threshold of zero for the SVM generalization pool (to form the template generalization). A higher assimilation threshold will allow separate clusters to form when the retrievals are not sufficiently similar. Since the unlabeled example is assimilated first and the retrievals are assimilated in order of descending similarity to it, the first cluster to form could be used as the template generalization to train the SVM. The rest of the clusters could be discarded for not being structurally similar enough to the unlabeled example and its nearest neighbors to serve as trustworthy training data.

A rigid number of remindings is also an intrinsic issue for the KNN-Weighted classifier in some of the experiments here for low-to-mid-ranging assimilation thresholds, specifically the experiments with relatively few categories. In the Freeciv experiment, for example, there are six categories. In a sample run with an assimilation threshold of 0.3, the resulting generalization pools, shown in Figure 5.12, together only contain a total of nine prototypes. Using a weighted KNN classifier with k=9 (or higher) in this case guarantees that all prototypes factor into every classification decision. Some categories ended up with a single prototype each (*bay*, *isthmus* and *strait*). Depending on the variety in the vocabulary used for encoding, there is a base level of similarity that can be expected even between structurally dissimilar examples, because SME will take the opportunity to match some individual fragments, unattached to larger structures, by virtue of their sharing an attribute or relation and not conflicting with other fragments. This is related to the *what goes with nothing* problem from Section 3.3.2.1. The higher the base level of similarity, the more the single-prototype categories will be systematically disadvantaged, because having more prototypes contribute similarity scores to the classification decision will allow other categories to overtake even a highly similar comparison to a single-prototype category. This is even more true when using an unweighted

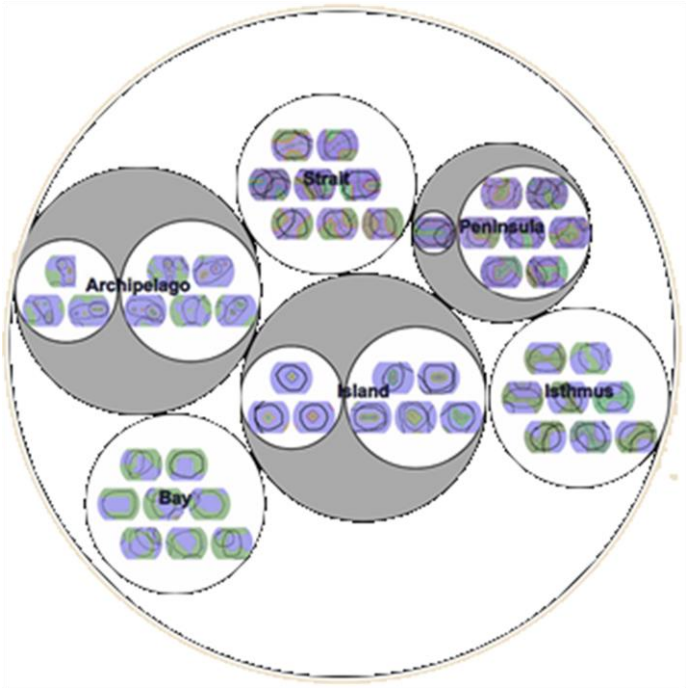


Figure 5.12: The generalization pools resulting from a run of the Freeciv dataset, with a low assimilation threshold of 0.3.

KNN. Such a systematic advantage is counter-intuitive, since a more compressed category can arguably be viewed as better understood, owing to an encoding scheme that better captures the unifying properties of examples in the category. ABLe, and SVM-KNN's in general, are resistant to these systematic preferences because one example from a category may be the only support vector for the category, obviating other prototypes from the category.

6 Related Work

6.1 Machine Learning

6.1.1 Structure Mapping and Machine Learning

Broadly speaking, Structure Mapping has been applied to classification in two different ways. First, there have been approaches that ultimately decided a label based on similarity to training cases, either compressed using SAGE or not. This includes approaches where similarity scores are computed exhaustively (Dehghani and Lovett 2006; Lovett et al. 2006), or where MAC/FAC is used for scalability (Chen et al. 2019; McLure et al. 2011).

The other type of structure-mapping-based classifiers are those which perform statistical relational learning using the correspondences (and candidate inferences) output by SME. We touched on two notable examples in Section 5.1: Halstead's Bayes nets built on SEQL, SAGE's predecessor (Halstead 2011), and Liang & Forbus's SLogAn, an approach to structured logistic regression built on SAGE. In both cases, like in ABLe, analogical generalization is used to form a structured template than can generate a feature vector for every case that has been assimilated into it. This is a form of *propositionalization*, the extraction of a flattened feature set from structured or relational data.

One difference between the present work and Halstead's approach and SLogAn is the specific feature-based technique that was applied. Halstead used SEQL (Kuehne et al. 2000), a predecessor to SAGE, by adding all examples, regardless of label, to a single generalization pool. As in all three of these approaches, the generalizations and ungeneralized examples in the pool each constituted a feature template. For each generalization that forms in the pool, a joint probability table for all of the assertions in the generalization was maintained to perform

Bayesian inference. That approach seamlessly extends to supervised, semi-supervised, or unsupervised learning, and allows for sophisticated probabilistic inference. Only one generalization pool needed to be maintained for learning an arbitrary number of concepts at once. The joint probability table for each generalization could explode in size as it grew, but Halstead used an efficient data-structure to store the sufficient statistics for this table, ameliorating the scaling concerns. Halstead also used heuristics and meta-knowledge to override the default propositionalization method for some generalized assertions. Instead of using binary features reflecting the existence of an analog (*existential* propositions), some facts with numerical arguments were instead propositionalized with *characteristic* propositions – slots with numerical values. The SVMs in ABLe would be accepting of these types of features and may find them useful in sketch recognition.

SLogAn performed supervised learning. Separate SAGE generalization pools were maintained for each label, like the present approach. Positive examples of a label were added to the generalization pool for the label to construct probabilistic templates (generalizations and ungeneralized examples), and negative examples were assimilated into these templates afterward. Structured logistic regression was used to modify the weights in the templates to output high scores for positive examples and low scores for negatives. These weights capture the concept hypotheses. Here we incorporate discriminative techniques different from logistic regression – specifically, rule learning and support vector machines. Because support vector machines minimize the hinge loss whereas logistic regression minimizes the logistic loss, support vector machines are less sensitive to outliers.

The near-miss learner is similar to SLogAn in that analogical generalizations of positive examples form the prototypes in which hypotheses are stored. Comparisons with negatives are

the genesis of hypotheses in the near-miss learner, and the driving force for modifying the weights in the templates in Liang and Forbus's approach. In their approach, the prototypes are expanded to accommodate negative evidence by assimilating the negative examples into the prototypes after the positives have been assimilated. In the near-miss learner, the negative examples are not assimilated by generalization. Instead, exclusion hypotheses accommodate negative evidence. They are similar in that they directly refer to the elements in a prototype, together forming a single structured model with positive and negative evidence, but they are more constrained because they originate from candidate inferences, which are only generated for structure in the negative that mentions elements that have analogs in the positive (more distant structure that only shows up in the negatives does not play a role in the model like it does when negatives are assimilated by analogical generalization as in SLogAn). This loss of expressiveness comes with more natural scalability. For most concepts, the number of negative examples available either by observation or inference is much larger than the number of positives. One must be selective when assimilating negative examples into prototypes.

Similar to SlogAn and Halstead's work, ABLe uses analogical generalization to assimilate positives and negatives into the same template, but the template is ad hoc and of predetermined size to keep tight control over scalability. Unlike these other approaches, ABLe puts a singular focus on the unlabeled example, using it to determine which labeled examples will be assimilated into the template and the order in which they will be assimilated. This comes with hypotheses that are on-demand and ephemeral.

6.1.2 Inductive Logic Programming (ILP), propositionalization, and structured-data SVMs

Here we dive deeper into comparisons discussed in Section 5.1. Inductive Logic Programming (ILP) traditionally applies version space techniques within hypothesis spaces consisting of first-order logic programs. Given background knowledge as a set of definite clauses and a set of positive and negative examples in the form of ground literals, the basic formulation of ILP searches for a hypothesis – a set of definite clauses – that covers all of the positive and none of the negative examples.

Traditionally, the search for a hypothesis in this space involves advancing specific and general boundaries based on positive and negative examples (respectively) in order to prune the space. The specific boundary is moved (generalized) by removing literals from a clause and by anti-unification, which produces quantified variables. The general boundary is advanced by applying a refinement operator, which builds progressively more specific hypotheses by either adding literals to a clause or assigning ground terms to variables within the sentence (called θ-substitution). The search is heuristic and often greedy, based on gains in discriminative power. Near-miss hypotheses in our approach similarly advance a general boundary by adding necessary conditions that act like literals in a clause, but this general boundary is only defined locally in the similarity space, referring to the elements of a SAGE prototype (a generalization or ungeneralized example).

Learning disjunctive definitions is a known challenge for ILP systems because of an explosion in the hypothesis space, though several foundational systems allowed for disjunctions. INDUCE (Michalski, Carbonell, and Mitchell 1983), FOIL (Quinlan 1990), Progol (Muggleton 1995), and Aleph (Srinivasan 1999) construct disjunctive rules in a greedy fashion. FOIL, for

example, incrementally adds clauses to cover additional positive examples and incrementally refines each clause to cover none of the negatives. One issue with these approaches is that performance varies widely based on the order of the examples; the early conjunctions formed will tend to cover as many positive examples as possible without covering any negatives, while the later conjunctions formed will be targeted at the leftover positives. Another issue is that these approaches can get stuck in local maxima, because of the greedy search heuristics. Sometimes the most globally productive literal to add to a definition is one that exhibits low discriminatory power. GOLEM (Muggleton and Feng 1992) and FOIL adopted search heuristics that check for a known class of these *determinate literals* – i.e. literals that introduce new variables that only have one possible binding for each binding of the other (non-new) variables in the literal (see (Quinlan 1991) for a more thorough explanation). Because the introduction of determinate literals can lead to an infinite regress, part of the search space must be closed off with this approach.

The near-miss learning approach in Section 4 organizes disjunctive definitions according to the clusters found by SAGE (Figure 4.3). On the upside, this lack of flexibility with respect to disjunctions reduces order sensitivity and avoids the need for greedy search heuristics. On the downside, disjunctive hypotheses in the near-miss learner lack the expressiveness of disjunctive hypotheses in ILP because of the lack of control over how individual clauses cover training examples – in the near-miss learner there is essentially one clause per (similar) cluster. Because disjunctions are forced among dissimilar training examples, the near-miss learner is expected to be less efficient than ILP at learning categories based on universal rigid criteria that cover a structurally diverse set of examples, such as *regular polygon*. The opportunity to learn a single clause covering all examples is lost if those examples are not all sufficiently similar. An avenue

for extension may be strategies for detecting when clusters can be explained with some universal rule, or perhaps a more parsimonious alternative is to detect such situations as opportunities for rerepresentation and, in response, modify both the learned SAGE representations and the encoding scheme. ψ-abstraction (Section 6.2.2 below) is a promising method of doing this for categories like *regular polygon*, where the important structural property is a regularity over an arbitrarily long sequence. On the other end of the spectrum, the near-miss learner also lacks the expressiveness to learn a concept definition that is disjunctive *within* a cluster of similar examples. Unlike the near-miss learners in Section 4, ABLe can represent some disjunctive hypotheses among similar examples very naturally (Figure 5.5), but the XOR remains inaccessible.

The most essential difference between analogical learning and ILP is the presence of a mapping step, which determines *what goes with what* across examples based on systems of relationships. This establishes a shared model with constants that hypotheses can refer to, in lieu of variables that must be constrained. The bet is that there may be critical properties that apply to a particular part of an example (e.g. the dorsal fin of a shark), and that a reliable way to identify which is the relevant part in a given example is by the system of relationships in the example that surround that part (e.g. the spatial relationships between the dorsal fin and neighboring parts, the relationships between those parts and their neighbors, etc.). With ILP, this system of relationships must be made explicit in the hypothesis in the form of a set of literals. These literals, despite being important for identifying the relevant part, may not be very discriminative in isolation if they are common structural relationships (e.g. above), making them unlikely targets for greedy search heuristics. Furthermore, larger systems or relationships would typically require more search with ILP. In fact, general-to-specific ILP would prefer to find the

minimal subset of the system of relationships necessary for covering the training examples, since it is the simplest valid hypothesis. This is in stark contrast to the structure mapping approach, in which the richest shared systems of relationships drive the mappings (due to the systematicity bias). In summary, the analogical mapping step implicitly stores much of the relational scaffolding that must be made explicit in an ILP hypothesis, and while structure mapping prefers to find richer scaffolding, ILP prefers minimal scaffolding.

For the past two decades there has been a sustained interest in merging inductive logic programming with statistical classifiers. Some approaches learn a probabilistic graphical model whose structure derives from relational data, similar to Halstead. Since we are interested in sketched data and we do not assume input data with stroke order, undirected graphical models are perhaps the most relevant. Markov Logic Networks (Richardson and Domingos 2006), for example, learn weights for formulas (clauses) in a first order knowledge base by, given a set of constants, constructing a ground Markov network for learning and inference that contains a node for every possible grounding of every predicate and features for every possible grounding of every formula. The advantage is that learning and probabilistic inference techniques are well understood for Markov networks. However, this family of techniques has trouble scaling with the number of constants and the number of formulas in the knowledge base. To be flexible enough to be applied to different domains, a sufficiently expressive set of formulas must be represented (Kok and Domingos 2007), which makes scaling even more of a challenge. Like with traditional ILP, the set of formulas generally starts out simple and gets more complicated with more learning, making elusive the types of hypotheses that analogical learning picks up on naturally – those involving rich systems of relationships (Section 5.1).

ILP has been combined with support vector machines in a variety of ways. The common thread is the use of *propositionalization* – the flattening of relational representations into a feature set that is amenable to traditional machine learning classifiers. In practice, propositionalization techniques are often customized for a domain, though (Cumby and Roth 2003) provides two examples of general propositionalization frameworks, i.e. feature extraction languages, that apply to description logics. When propositionalization is not tailored to a domain, there is a trade-off between the expressiveness of the hypothesis space and a combinatorial explosion in the number of features, which can lead to reduced interpretability, overfitting, or intractable learning. In ABLe, SAGE is used to extract features from relational representations. It keeps the set of features relatively small by analogically compressing a limited number of similar examples into generalized model, which has a size bounded by the sum of the number of facts in these examples. Nonetheless, since the classifiers learned are linear, a feature extraction language like that of (Cumby and Roth 2003) could be useful for feature construction to deal with the XOR problem per the discussion in Section 5.2.1.1.

SVILP (Support Vector ILP; Muggleton et al. 2006) combines ILP and support vector machines using the *kernel trick*. The kernel trick can be used to map instances into a many-dimensional, potentially infinite-dimensional feature space, where the kernel is an inner product in that space. In this higher-dimensional space, a linear SVM can be trained. This works because the SVM algorithm only uses instances via their inner products. The mapping into the higher dimensional space does not need to be known to verify that the kernel is an inner product – it can be verified from the properties of the kernel matrix, specifically whether they satisfy Mercer's condition. SVILP defines an inner product between examples based on how they are classified by every hypothesis in the hypothesis space, which in typical ILP fashion, is defined

using first order logic in the context of background knowledge. Since the hypotheses in the hypothesis space are first-order rules and can incorporate background knowledge, the kernel can be used to classify examples based on sophisticated structural properties. Kernel-based SVMs are most powerful when the number of training examples is larger than the number of features in the original feature space, but not so large that the super-linear training complexity is prohibitive (on the order of 100,000 training examples). In ABLe, the number of training examples is small compared to the number of features because of the practical bottleneck of a number of SME comparisons that is linear to the number of training examples. In this setting, linear SVMs tend to perform well since there is plenty of dimensionality to separate the examples. Furthermore, an issue with kernelization is the lack of explainability, since the contour of the concept boundary cannot be easily interpreted in terms of the original feature space. Linear combinations of understandable dimensions are relatively intuitive (e.g. in telling sketches of bears from sketches of dogs, perhaps tail length is twice as influential as ear size).

There have been some notable approaches for exploiting the power of kernel methods while supporting explainability. kFOIL (Landwehr et al. 2010; De Raedt et al. 2006) combines the same notion used in SVILP – that the hypothesis space can be used to characterize an example with a set of features corresponding to whether a hypothesis covers an example - with the idea of *dynamic propositionalization* – learning a kernel as a relational hypothesis in the traditional ILP hypothesis space (a subset of the clauses in the space). The hypothesis that defines the kernel can be modified by starting with a very simple, general hypothesis and applying the same search strategies from traditional ILP. In the case of kFOIL, the FOIL refinement operator of adding literals to clauses is used. An interesting aspect of kFOIL is that the kernel can be learned for multiple tasks at once. The learned relational hypothesis, which determines the kernel, is

typically simple (few clauses), especially in the multi-task setting. Therefore, even though classification in the kernelized feature space is still obscured, the kernel itself carries more interpretable meaning than in static propositionalization approaches like SVILP, by virtue of the fact that it is associated with an explicit hypothesis in the relational hypothesis space that is, ideally, concise. This is still less explainable than a linear SVM because the similarity measure to measure instances is known, but the shape of the boundary and the locations of instances in the feature space is not.

Another relevant approach to combining the power of kernels and the explainability of linear separators is the SVMrule method (Navia-Vázquez and Parrado-Hernández 2006). There, a semi-parametric kernel-based SVM is used to learn a simplified nonlinear boundary with (controllably) few support vectors, and then a Voronoi diagram between the resulting "prototypes" in the higher dimensional space is used to split the training data into regions near the boundary. (The Voronoi diagram is computable in this space because it is based on distance.) Inside each Voronoi cell, a linear classifier is trained on the local training data. The idea is to piece-wise approximate the non-linear boundary with linear classifiers, with a controllable level of granularity. Their approach makes classifications locally explainable, where locality is based on a similarity measure – a philosophy in line with Structure Mapping and our approaches in this work.

The idea of partitioning training instances by similarity and then learning simpler local classifiers shows up elsewhere. Cleuziou et al. (2003) outline an approach to learning disjunctive concepts that is perhaps the closest ILP analog to our near-miss-based approaches. First, positive examples are clustered into *subconcepts* according to a similarity measure for first order logics that is similar to the inner-product used in SVILP. Then, a separate single-clause

hypothesis is learned for each subconcept based on the negative examples. This approach mitigates the issues that face ILP systems that greedily search a disjunctive hypothesis space – namely the overwhelming size of the hypothesis space and sensitivity to example order – because non-disjunctive definitions are constructed for clusters in isolation. The k-Means+ID3 method (Gaddam, Phoha, and Balagani 2007) is an example of this same general strategy being applied to generic feature-vector representations.

Finally, we note that the linear SVM is not the only approach to maximum-margin classification with a linear model. Large Margin Nearest Neighbor (LMNN; Weinberger and Saul 2009) classification uses hinge loss (like the SVM) to optimize a linear transformation of a k-nearest-neighbors distance metric expressed as a Euclidean norm, in order to maximize the margin between same-label neighbors and different-label neighbors. In other words, the linear model is trained to amplify or suppress differences along particular dimensions in such a way that simultaneously pulls same-labeled neighbors (of the k nearest) closer together and pushes differently labeled neighbors (of the k nearest) farther apart in Euclidean space. Their approach has the appealing property that it extends naturally to multi-label tasks, whereas the SVMs find binary decision boundaries and must be combined to perform multiclass labeling, which is arguably less explainable because of the voting method of synthesizing the binary judgments and because there are *n* (one-vs-all) or $\frac{n(n-1)}{2}$ (one-vs-one) linear models to understand. It is also an interesting approach in that it is optimizing for local distances from neighbors. First of all, this could be a simple, explainable way to fit the MAC computation (a dot product in Euclidean space) in MAC/FAC to a particular domain or training set, and/or to a particular *k*-value. MAC is very influential in our approaches. In Retrieve-KNNW, MAC completely determines the set

of k neighbors, while SME determines the weights among those neighbors. In ABLe, MAC completely determines the subset of instances, or k neighbors, that the SVM will train on. LMNN could optimize this choosing of k in a way that is highly explainable – simply a set of weights reflecting the importance of each predicate in determining (MAC) similarity. Second, the local nature of the optimization in LMNN would allow ABLe to accommodate disjunctive clusters of similar same-label examples even among the instances that make it to the SVM stage, i.e. those that are retrieved and assimilated into the template generalization. This is likely overkill when the number of instances in this stage is far outnumbered by the number of dimensions (assertions in the template), but it could be very useful if ABLe were extended using the *entity-mapping network* idea in Section 7.3.1 below.

6.2 Sketch Recognition

There have been other approaches to classifying sketched symbols and objects that use symbolic descriptions as intermediate representations. The distinction between sketched symbols and sketched objects is a matter of whether the classes being sketched are members of a standardized, simplified vocabulary or an arbitrary vocabulary for which renderings are largely subjective. Sketched symbols are used in tightly constrained diagrammatic domains (e.g. circuit design, architectural floor plans) whereas sketched objects are relevant to less constrained design domains (e.g. interior design) or ecological renderings (e.g. a food-web). While there has been plenty of research on sketched symbol recognition that has converged on near-perfect performance within some narrow domains (Ouyang and Davis 2011), the research on sketched object recognition has been relatively sparse. This section begins with a discussion of some

related work on sketched symbol recognition and transitions into examples of sketched object recognition.

LADDER (Hammond & Davis 2004; 2006) is a multi-domain language for describing sketched symbols. The language expresses geometric constraints on the primitives that comprise a symbol, such as connectivity, relative length, and relative orientation of edges. One difference between LADDER descriptions and our prototypes is that LADDER specifies constraints on variables, which must be satisfied in order to positively recognize a shape. All possible variable assignments are tested and constraints are hard (strictly enforced), whereas the approach proposed here effectively learns both hard and soft constraints that, for the most part, do not contain open variables. While LADDER is extensible to a variety of complex shapes in theory, in practice it had trouble with shapes containing many curves, or in which curves are defining features. LADDER can actively learn via automatic generation of potential near-miss sketches, which is a main target of our approach, motivated in the beginning of Section 4 and returned to in Section 7.3.10.

Gross (1996) described the digital cocktail napkin, which recognized a sketched symbol by inscribing a 3x3 grid into its bounding box. Salient features of the ink are described for each cell in the grid. The grid approach provides invariance to non-uniform scaling and optionally to reflection and rotation in increments of 90 degrees. However, this approach is vulnerable to fine-grained variations due to the coarse-grained grid.

Lee et al. (2007) performed symbol recognition by matching Attributed Relational Graphs (ARGs). Hand-drawn symbols were carved into primitives, and like in the approach here, relationships and attributes of these primitives were used to construct a graph. Graph matching allowed “average” descriptions to be abstracted across multiple training examples, but these

abstract descriptions retained the most common descriptors for each node rather than keeping a distribution of descriptors as in SAGE. The authors compared accuracy and resource cost (classification time) for multiple graph matching algorithms and found that a greedy approach achieved the best trade-off between these two metrics.

Lladós et al. (2001) performed sketched symbol recognition via graph matching on Region Adjacency Graphs (RAGs). Rather than representing relationships between and attributes of ink primitives such as lines and arcs, their approach does so for regions (minimal closed loops of lines and arcs) much like our atomic edge-cycles. Whereas our approach used a similarity metric based on structure mapping theory, Llados et al. used an edit-distance measure between RAGs. Their approach was capable of identifying multiple instances of learned objects within an unlabeled input scene – a more complex recognition task to which the approaches here must be extended to perform the duties of a full-fledged sketch collaborator.

A particularly similar perceptual organization approach to ours in is Sanscribe (Saund et al. 2002), which likewise carves hand-drawn line art into curvilinear stroke fragments called prime

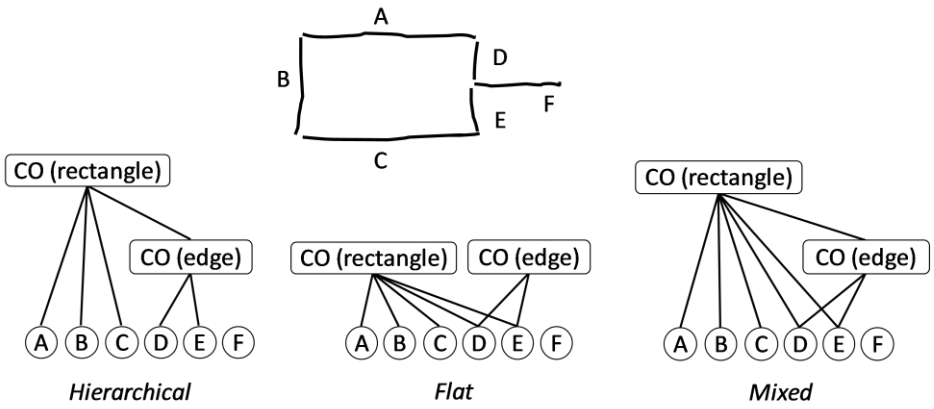


Figure 6.1: Taking an example from (Saund et al. 2002), at the top is a set of five ink fragments from a box-and-line diagram. Along the bottom are three alternative approaches to assigning membership between the perceptual entities.

edges (analogous to atomic edges) whose endpoints are clustered into corners. To find composite objects, paths are traced through these edges using closure (analogous to atomic edge-cycles), continuity (analogous to the union of continuity edge-cycles and super-edges), and a third, stroke-based rule that traverses fragments that are part of the same stroke even if they contain a salient curvature discontinuity. Like in CogSketch, a structured representation of these perceptual entities is produced by Sanscribe. Among approaches with this objective, Saund et al. distinguish between those that construct a hierarchical structured representation (Figure 6.1, left), and those, like Sanscribe, that produce a flat structured representation (Figure 6.1, center; not to be confused with propositionalized, or feature-based, representations). The distinction is illustrated in for an excerpt portion of a box-and-line diagram. Our approach combines these alternative views into a mixed structured representation (Figure 6.1, right), which allows analogical generalization to more naturally accommodate statistics about both views of the structure in the same generalization, at the cost of larger representations. Representational bloat is why our approach did not produce composite edges using the third rule above, but selectively adding composite edges that ignore discontinuities could potentially make our analogies more robust to false-positive (unintended) corners. Notably, the Sanscribe system tackled a high-impact problem in sketch understanding that we ignore – the early segmentation of written text from line art as a first step in perceptual organization, to process them differently. CogSketch sidesteps this challenge by providing a structured interface for annotations.

The authors that introduced the Berlin dataset also used it to evaluate several sketched object classifiers that used a bag-of-words representation (Eitz et al. 2012), a popular approach in computer vision. The best performing of their classifiers, with 56% accuracy, used a multiclass kernelized SVM, where the kernelized distance between the two histograms (bags of words)

representing two sketches is based on the soft-assignment of local image descriptors to a vocabulary of visual "words" (clusters of local image descriptors from pretraining) that had been extracted from a subset of the training set by clustering local image descriptors. The bag of words approach has inherently limited explainability because it encodes the frequencies of local visual structures but nothing about overall structure. The visual words can be visualized, but Eitz et al. used 500 of them – not an unusual amount, but for a human, a potentially overwhelming set of non-symbolic signals working in concert. Nonetheless, Eitz et al. (2012) demonstrated that the representation at least supported finding representative examples of a category (multiple if the category clusters in the feature space) and visualizing the distribution of instances in the feature space by dimensionality reduction.

Schneider and Tuytelaars (2014) achieved 68.9% accuracy on the TU Berlin dataset using a Fisher kernel, which takes a generative model and measures the distance between instances by how they deviate from the model. The generative model in their case was a Gaussian Mixture Model with 256 Gaussians, with its parameters estimated by sampling local features from the training set. The image representation that performed best was a spatial pyramid of SIFT patches, which are a popular type of feature used in computer vision. Because sketches are visually sparse, uncommonly large SIFT patches were found to perform better, which is consistent with the custom features in Eitz et al. (2012). This approach is notable for its state-of-the-art performance achieved using standard computer vision representations and discriminators. While the learned model itself is opaque, they were at least able to quantify the importance of strokes – separating informative ink strokes from confusing strokes – by observing the impact on classification of removing the stroke.

Convolutional neural networks' dramatic success in computer vision has extended to sketched object recognition. Sketch-a-net (Yu et al. 2015) was famously the first approach to outperform humans on classifying the full TU Berlin dataset. The comparison was not entirely fair because Sketch-a-net used the original stroke information (segmentation and ordering) as a core organizational principle whereas humans could only look at the finished sketch, rendered in pixels, as would be the case when recognizing pen-and-paper sketches. Nonetheless, Sketch-a-net's break-out success is still important because more and more sketches are being generated in digital ink.

The sketch-rnn (Ha and Eck 2017), a recurrent neural network, is a generative model for digital ink (with stroke information). It is capable of learning to draw concepts from scratch or by continuing a partially drawn sketch. Under certain conditions, the same latent vector embedding that supports continuation also supports conceptual combination by allowing the model to perform drawing arithmetic, e.g. the difference between a pig and a pig's face – the body – can be added to a cat's face. This arithmetic acts as if it is exploiting analogies embedded in the latent space, projecting the difference between two similar but non-identical inputs onto a third. Unlike the approach here, where the analogical mapping process and the input and output representations are symbolic and auditable, the internal representions of the sketch-rnn are a black box that can only be indirectly understood by observing its outputs given its inputs. This work also produced the QuickDraw dataset, which is an attractive corpus for future development and evaluation because the online game used to gather the sketches strongly incentivized fast, minimalist, communication-driven drawings.

6.2.1 Bayesian Program Learning

The Bayesian program learning (BPL) framework (Lake, Salakhutdinov, and Tenenbaum 2015) achieved human-level performance on a one-shot learning task in which images of hand-drawn characters from a novel alphabet were classified after having observed one example image of each character in the same alphabet drawn by someone else. The system had been pretrained on 30 different hand-drawn novel alphabets that included stroke information (including stroke order and timing information to detect pauses, as opposed to images), in order to learn a library of primitive parts of strokes that could be used to compose examples, and to learn statistics about the how these parts are combined to form symbols, or *tokens*.

Equipped with these priors from pretraining, BPL treats the construction of a token as a generative causal process that composes a motor program by sampling (1) a number of strokes, (2) a number of parts for each stroke, (3) a sequence of specific parts for each stroke based on the number of parts and the ordering statistics from pretraining, and (4) a sequence of topological relations to coarsely determine how each stroke attaches to the previous stroke in the sequence. Drawing a token from this type-level motor program first involves sampling, for each part in sequence, a start location for the part based on the qualitative topological relation in the program, and a degree of motor variance, together determining a (quantitative) trajectory for the part. Finally, an affine transformation for the whole image is sampled and then an image is sampled based on the transformation and the part trajectories.

During one-shot learning, the BPL infers which character an unlabeled image corresponds to from an alphabet provided. There is only one available labeled example of each character in the alphabet. This posterior inference involves searching the space of possible motor programs for

ones that can explain (match) both the unlabeled image and one of the labeled images. This space is combinatorially explosive even in their relatively constrained symbol recognition task and the authors found that Markov chain Monte Carlo (MCMC) sampling was too slow and vulnerable to local minima in its search. Instead, they approximated the posterior distribution by searching for good candidate parses (in terms of strokes and stroke parts) based on bottom-up visual cues and then performing MCMC in these regions of the search space. This bottom-up visual processing involved extracting from the ink an edge/junction graph (analogous to the one used in this work), which was meant to capture the *skeleton* of the ink (analogous to how we capture the skeletons of the MNIST digits in Section 2.3.5). Because BPL expressed its motor programs in terms of strokes and parts of strokes, candidate (stroke) parses were determined from the skeleton graph by sampling random walks in the graph, tracing from each ink segment to the next with probability based on a continuity measure.

The BPL framework outperformed state-of-the-art deep learning algorithms on the one-shot symbol recognition task, given the same background data for pretraining. Its human-level performance after so few examples distinguished it from these alternatives and makes it an especially interesting related approach here, given the learning priorities we set out for sketching collaborators (Section 4). It is this author's interpretation that BPL's success owed in large part to its modeling of a drawing motor program as a distribution over compositions of intermediate parts joined with qualitative relationships, as well as its use of bottom-up image cues to search for candidate compositions during inference. BPL's causal organization of ink into ordered strokes (as opposed to unordered edges), critical in its impressive Turing-tested ability to generate novel examples, was perhaps also a useful learning bias for hand-drawn characters because they are often drawn in the same order by different people due to the natural affordances

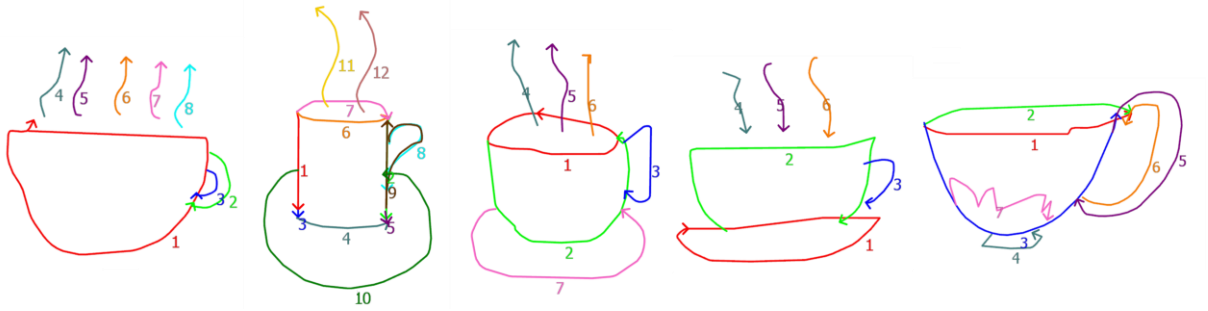


Figure 6.2: Five examples of the concept Cup from the TU Berlin dataset , with strokes separated by color, numbered to reflect stroke order, and given arrow heads to reflect stroke direction.

of the shape and the direction of writing (e.g. left to right). Sketched objects are drawn with more variability. For example, despite being a relatively similar set among the examples of *cup*, the cups in Figure 6.2 have very different stroke orders. Unlike most characters, sketched objects can also vary in the intended visible parts of examples (e.g. whether a cup is drawn with a handle, with a saucer, with steam rising off the top, or with the contents visible) and the intended depiction of these parts (e.g. an opaque, two-stroke handle vs. a thin, single-stroke handle). For sketched object recognition, perhaps an approach like BPL's could be used to learn more abstract causal models for generating drawing programs in terms of functional parts at a high level, surfaces and shapes at an intermediate level, and strokes or stroke primitives at the lowest level. Of course, these added layers would further explode the combinatorial search space for candidate parses, and likely expand the need for bottom-up cues for composing these candidate parses. The bottom-up perceptual organization methods extending edges to edge-cycles in Section 3 may be relevant in this regard, and bottom-up methods for inferring more complex parts from these elements could be a useful extension for both approaches.

Whether organizing by strokes or higher level, the success of BPL points to the effectiveness of learning to recognize sketched instances using a causal model containing latent variables that

reflect the *process* of sketching. To this end, Halstead's probabilistic models learned from analogical generalization may illuminate avenues of future work for the analogical approach here.

6.2.2 Tractably Encoding Sketches for Analogy

Some sketched objects contain variable-size groups (sequences, cycles or clusters) of visual entities that are highly regular with respect to attributes and local relationships. For typical structure mapping approaches, which rely on one-to-one mappings between the elements in pairs of cases, repetitive structures pose several issues:

1. For differently sized groups in base and target, there will be leftover entities that are potentially co-opted to participate in spurious correspondences, a la the *what goes with nothing* problem introduced in Section 3.3.2.1.
2. The selection of which entities are leftover may be unstable across different generalizations and comparisons but may be progressively reinforced within a given generalization (because fact probabilities weigh more heavily in the mapping). This can lead to artificially divergent prototypes.
3. Groups may be intuitively interpreted as a single part but will nonetheless have an impact on the mapping that is roughly proportional to their cardinality, which may be arbitrarily large.
4. Salient properties of the group as a whole – e.g. the elliptical perimeter of a turtle shell texture and its connectivity with the limbs of the turtle – will be obfuscated.
5. Patterns of relationships between tuples of entities in the group that help define the group by induction will not be detected or propositionalized.

6. They risk making analogical matching intractable with respect to time.
7. They risk making SAGE generalizations intractable with respect to long-term storage.

Our inclusion of continuity-based edge-cycles in the edge-cycle representations in Section 3 ameliorates issue #4, but only in cases when the group is circumscribed by a relatively continuous shape.

ψ-abstraction (Friedman 2015) is a way to encode sequences and cycles of entities in a relational graph in a succinct way. It compresses isomorphic subgraphs that are connected along a path within a graphical representation of a relational case, such as the representations we use here. These paths are detected based on some repeating relationship in the relational graph and are compressed by feeding the *relevant subgraph* of each entity in the sequence into SAGE. ψ-abstraction uses the resulting analogical generalization of these subgraphs to build an *isomatrix* that describes the sequence or cycle in tabular form. The path and isomatrix are used to identify the *current term*, and optionally the *previous term*, and *next term*, which, together with their relations to one another and other entities in the abstraction, represent an inductive definition of the path. The path can be reified in the original graph, and the repeating substructures along the path can be replaced with the single generalized substructure found by ψ-abstraction, which is related to the path entity. Other summary knowledge about the path is encoded, including the qualitative characteristics of quantitative sequences in the isomatrix (e.g. a functional value of the current term is decreasing monotone). The sequence of entities in the path are also kept around, stored in an extensional sequence representation that is associated with the reified path. At least for sequences and cycles, ψ-abstraction directly addresses issues #2-7. It potentially reduces the risk posed by the *what goes with nothing* problem (#1) because even though the original entities along the path are kept around in the representation, as much as possible of their

nearby relational structure has been stripped away in favor of generalized relational structure that is local to the generalized entities instead.

Note that ψ-abstraction operates entirely at the knowledge level, making it a useful rerepresentation technique for the same reason as the filters in Section 3.3 – it can be applied to instances stored in long term memory after the original information source (e.g. a sketch) is no longer available. It might be able to support an offline process that searches for opportunities to compress representations while strengthening their mapping to other representations, especially those of the same label. Unlike our filters, it can be used to compress the representations stored in long term memory in such a way that can be partially reversed. ψ-abstraction could be used in conjunction with the entity-based filters in Section 3, since it retains enough information to (a) determine which atomic entities the path grounds out in, thanks to the extensional list of member entities, and (b) approximate the sizes of any atomic elements that are on the path.

Some groups of repetitive entities form clusters of proximal entities (e.g. the cells in a turtle shell or the spokes in a bicycle wheel) or fields of regularly distributed entities (e.g. raindrops or pizza toppings). The texture detection technique in (McLure et al. 2015b) is designed to deal with the first type. Unlike ψ-abstraction which operates on the knowledge level, this texture detection technique operates on the quantitative ink, which is useful because for some highly textured objects even the computation and storage of the full edge-cycle representation is intractable. This technique uses the Ising model, an undirected graphical model that makes two assumptions: (1) The node labels are binary, and (2) every edge energy can be expressed as a disagreement cost – a fixed energy cost incurred only when the labels for the two nodes connected by the edge have different labels. The approach capitalizes on the planarity of our atomic edge-graphs to build a planar Ising model at the edge-cycle level, which can be solved

efficiently even with real-valued disagreement costs between neighboring edge-cycles. Using real-valued disagreement costs allows for a completely bottom-up grouping, because using positive disagreement costs in the Ising model (for submodularity) necessitates background knowledge that strategically and correctly biases particular elements in the Ising model to either group or not, which is not trivial to provide. The disagreement costs between adjacent edge-cycles were sums of components measuring different aspects of the local geometry of where neighboring edge-cycles touched, but in future work the weights on these components should be learned, either from pretraining or by hill climbing on each sketch separately with a target function that rewards texture regions with geometrically simple perimeters. Another possibility would be hill-climbing with the overall propensity to group in order to target a specific number of entities. These potential extensions are why the method is concerned with guaranteeing efficient solutions – it was intended to be called iteratively in a search process. This method may be extended to edges or to fields of disconnected edge-cycles by using the Delaunay triangulation (a planar graph) on the centroids of elements. Sufficient texture detection will be an important ingredient for tractably running on the full TU Berlin dataset.

Lovett, Dehgnani, and Forbus (2007) classified sketched objects using MAC/FAC. CogSketch's edge-based representations were used, with a previous generation of the edge-segmentation algorithm. The representations were filtered there as well to remain tractable and avoid swamping the match with too much detail. In their approach, assertions were filtered with a method that factored in entity properties as well as predicate properties. Edges further toward the exterior of a shape were prioritized. Attributes and certain high-order relations – termed *anchor relations* for their intended purpose in anchoring the match – were prioritized over common pair-wise relations. Assertions were ranked first by predicate preference, then by entity

preference. Their experiment using the Sun Up to Sun Down (Buckley 1979) dataset found an increase in performance between a case size of 100 and 150 assertions, but no significant differences between 150, 175 and 225 assertions per case. Their approach contributed a simple hybrid method for combining entity filtering and predicate filtering and alternative prioritization schemes from those in Section 0 for both entity-based and predicate-based filtering. Their preference for exterior edges is similar in spirit to our area-coverage and ink-coverage entity-based filters over representations with edge-cycles. Theirs was intended to capture the exterior shape of an object first and foremost. Perimeters have better area coverage than every other edge-cycle they contain, and usually better ink coverage. Our entity filter often moves next to large continuity cycles or regions of whitespace to capture the largest embedded shapes, whereas Lovett et al. (2007) continue in from the perimeter by following edge connections.

6.3 Categories and Classification in Humans

Smith & Medin (1981) pointed out three distinct views of categories that humans tend to acquire. The classical, or theory view of categories is that they are defined by logical criteria (e.g. triangle, grandfather), which may be tested against unlabeled examples in order to classify them. The probabilistic, or prototype view describes categories using characteristic descriptions that capture their typical properties (e.g. chair, cup). Prototypes for categories are often, but not always[4], based on the *central tendency* of the observed instances of the concept, i.e. the mental average of their properties (Solomon, Medin, and Lynch 1999). A third view – the example view – does not attempt to construct a unified representation of a category, but rather describes a

[4] Prototypes for goal-directed concepts (e.g. *clothes to wear in the show*) and some domain concepts of domain experts have been shown to be based on ideals rather than central tendency (Solomon et al, 1999).

category in terms of its specific instances (e.g. suicidal psychiatric patients). In the prototype and example views, unlabeled examples are classified by measuring their similarity to one or more of the known prototypes/example.

A given category may be treated from different views depending on the process that is operating on it. Osherson & Smith (1981) argued that while the prototype view is a compelling explanation for how people make fast membership judgments for a category, it cannot by itself explain concept combination (e.g. inferring a prototype for *pet fish* based on the prototypes for *pet* and *fish*) or truth conditions (e.g. all A's are B's), which can also be seen as links between concepts. They proposed a hybrid model in which each concept consists of both an *identification procedure* and a *core*. The core stores the definitional features of a category, facilitates conceptual combination, and allows truth conditions to be evaluated, while the identification procedure stores information relevant to rapid membership judgments. In support of a hybrid view, Armstrong et al. (1983) found that humans can distinguish between judging something as an instance of a concept versus a good or bad instance. Schwartz & Reisberg (1991) argue for a three-part hybrid view of concepts. In this model, each concept is associated with a prototype, a set of salient examples, and additional abstract information, all of which work together for the purpose of inference.

The Heterogeneity Hypothesis (Machery 2009) goes further than hybrid theories in separating prototypes from theories. It says that while prototypes and theories can refer to the same lexical classes, such as "water", they are not intrinsically linked as dual parts of a water concept; they are recalled and applied in isolation. Machery identifies two conditions that hybrid concept views satisfy and the heterogeneity hypothesis does not: (1) *Connection* states that retrieving either the prototype or theory part of a concept enables the other part to be retrieved as

well (2) *Coordination* states that in cases when the prototype and theory parts of a concept contradict one another, one of the parts ends up winning out. The argument against a hybrid model claims that coordination is not always apparent in humans because in some scenarios they can be made to affirm contradictory statements about categories.

The near-miss techniques in Chapter 3 take a view closest to that advocated by Schwartz and Reisberg. SAGE tells an elegant story about how prototypes and salient examples can form and coexist, and near-miss learning extends that story to build theories on top of prototypes and salient examples. The ABLe approach does not learn theories per se, but it does go beyond similarity to prototypes and examples for classification by looking for informative properties with which to discriminate.

7 Conclusions and Future Work

In this work we made progress on multiple fronts toward more useful analogical learning from spatial input. The investigations were specifically concerned with encoding strategies for sketched input, the utility of discriminative analogies, and analogical learning approaches that are controllably tractable.

Analogical learning is promising for creating sketching collaborators because it is designed to extract from examples a set of inspectable, structured visual models for a category that are useful as quickly as possible, both for recognizing future examples and generating new examples. This is important for sketching applications because sketchable concepts can take such a wide array of visual forms, are often specific to persons and tasks.

In working towards sketching collaborators, this work grappled with foundational questions that unfortunately presented barriers to tackling the more glamorous questions of interest to the author. In our approach to the overall problem, questions about tractable discriminative analogical learning and effective internal representations seemed to precede, for example, questions about how best to probe those conceptual boundaries learned, or how exactly to generate original sketches from those internal representations. This work ended up addressing many questions tied to the same theme – how to most effectively apply analogical learning to sketched input under resource constraints. The lessons learned can serve future work that requires analogical learning at scale, with special relevance toward long-lived sketching collaborators.

This section revisits our claims and contributions while reviewing the evidence gleaned from this thesis work. We then try to synthesize some overarching lessons and suggest some of the most exciting avenues of future work toward sketching collaborators.

7.1 Claims revisited

i. For sketched object recognition by analogy, cycles of edges, extracted based on closure and continuity, are a more effective and efficient level of qualitative representation than edges.

We saw in Experiment 1 that an edge-cycle-based encoding outperformed an edge-based encoding for classifying sketched objects. The edge-based encoding was the best-performing version from (Lovett et al. 2006), in which the raw edge representations had been strategically filtered down to a size of 175 facts per case, while the edge-cycle-based encoding was used in its entirety. Despite a lack of filtering and size optimization, edge-cycles outperformed edges at each one's optimal assimilation threshold for analogical generalization, which determines compression rate during training, and for a range of similarity thresholds. Edge-cycle-based learning was shown to be robust to compression during training, down to an assimilation threshold of 0.4 (from the range 0-1).

In Experiment 2, on a larger dataset of sketched objects (both by number of categories and number of examples), edge-cycles outperformed edges dramatically for all tested combinations of filtering strategies and case size-limits tested. Meanwhile in all these conditions, using edge-cycles caused analogical matching to take less than half the time on average. The edge-cycle representations were only 50-75% as big as the edge representations (they were truncated by the case size-limit less often).

ii. Sketched object recognition by analogy can be improved by using a case filter on the input representations that greedily attends to the perceptual entities that represent the biggest portions of the least-represented ink.

iii. For sketched object recognition by analogy, with an entity filter based on ink/area coverage, attending to edge-cycles and edges together, with representational tissue connecting them, is more effective than attending to either one exclusively when controlling for maximum case size.

In Experiment 2, we tested a hybrid encoding scheme that combined the unfiltered edge and edge-cycle representations and added some topological representations to describe their connectivity. This connective tissue served two purposes. First, it was designed to encourage the matcher to come up with mappings at the edge and edge-cycle level that were consistent with one another in terms of how edges are grouped into cycles (or else the unmapped connective tissue would drive down similarity and introduce more candidate inferences).

Second, with qualitative representations that capture how edge-cycles (and super-edges) break down into atomic edges, we had a means for localizing the overlap in the ink that's characterized by any set of entities. Combined with the qualitative representations that estimate relative size, we had a coarse means for progressively attending to the entities in a sketch that, together, describe as much of the ink as possible, and then delve into detail in a breadth-first manner as case size-limit allows. This filtering strategy was dubbed the *ink-coverage* filter. Since the edge-cycle representation alone does not have edge-level information for estimating ink coverage, for that encoding scheme we can instead use containment relationships to decompose all entities into their atomic cycles, in order to estimate an entity's *area-coverage.* Importantly, since everything needed for these entity-based schemes is in the qualitative

representation, a filter with these properties can be applied to cases in long-term memory without having access to the original (quantitative) sketch. It can even be applied to a generalization that may not have a quantitative representation. Testing it on generalizations is left for future work.

When entity-based filters were applied to edge, edge-cycle, and hybrid encoding schemes, the hybrid scheme outperformed for all case sizes. The entity-based schemes outperformed a predicate-based filter based on predicate rarity. The accuracy of the entity-based filters on representations containing edge-cycles (hybrid & edge-cycles alone) progressively decreased as the case size-limit rose from 100 to 300, indicating that the entity-filter, on edge-cycle-based representations, was not only more tractable but more effective than the unfiltered representations (since filtered and unfiltered are equivalent when the limit exceeds the maximum case size). The accuracy of the predicate-based filter, on the other hand, increased as case size-limit increased, indicating that the filter was a detriment to accuracy compared to the unfiltered representations. The hybrid scheme with the predicate-based filter, for all size-limits, performed worse than edge-cycles alone for all filters and size limits. Juxtaposed with its outstanding performance using an entity-based filter, it was evident that the strength of the hybrid scheme was not necessarily due to mapping edges and cycles at once, but was at least in part due to the availability of informative edges to augment the edge-cycle representation, exploitable by an effective entity-selection strategy.

iv. Criteria learned from near-miss comparisons and refined by analogical generalization can improve the performance of a similarity-based classifier by censoring retrievals that violate a hypothesized criterion, but the benefit is not consistent when these near-misses are between analogical generalizations themselves – an approach with more plausible storage requirements.

v. Extending a similarity-based classifier with hypothesized criteria from near-misses can improve its performance when the hypotheses are generated in moderation, which is not the case when detecting near-misses between analogical generalizations at relatively low similarity thresholds.

A common approach to using structure mapping for classification has been to use the structural similarity score returned by SME, in a nearest neighbor or approximate nearest neighbor framework. In Experiment 3 we tested approaches that used comparisons between training cases to hypothesize necessary conditions, and then apply those hypotheses in an otherwise similarity-based (nearest neighbor) classifier to censor similar examples that violate necessary conditions. The comparisons between training cases that yielded criteria were near-misses, and only formed when the similarity score of the comparison exceeded some similarity threshold. One of the two near-miss based classifiers in Experiment 3 outperformed similarity-based classification consistently, but that approach does not scale because the amount of storage needed is super-linear in the number of training cases. The other near-miss based classifier detected near-misses between prototypes formed by SAGE, which allowed for sub-linear storage requirements (due to SAGE compression). This approach showed some evidence of outperforming the similarity-based approach for high similarity thresholds (light compression), but the difference was only significant on one of the two tasks. For lower similarity thresholds (more aggressive compression and near-miss detection), the near-miss hypotheses – or more accurately, the near- *and* far-miss hypotheses – severely harmed this more tractable approach compared to the similarity-based baseline. At this aggressive level of compression, the simple nearest-neighbor classifier is more effective. In fact, its performance was very consistent across

the whole range of assimilation thresholds tested, in which the lowest threshold compressed the training data to 10% of its size (in facts).

One relationship that correlated with the relative performance of two near-miss classifiers was how many criteria hypotheses were left at the end of training. The nearest-neighbor baseline allowed us to infer that the quantity/quality of the hypotheses (rather than the quality of the retrievals at testing time) were likely responsible for improving performance in moderation and hurting performance in excess. For the same similarity threshold, more near-misses were detected between prototypes (in the scalable version) than between individual examples (in the more accurate version), in so far as there were near-miss candidates available. But even when the more scalable version was producing fewer near-misses because compression was leaving it with fewer candidates, it maintained its far greater number of hypotheses. Each near-miss comparison in these conditions was yielding a flood of noisy criteria hypotheses.

It was evident that near-misses containing fewer differences could be exploited via analogical comparison to the benefit of a similarity-based classifier. It is likely that our coupling of the near-miss threshold and the similarity threshold, while more parsimonious, obfuscated the benefits of other near-miss policies (e.g. having a higher bar of similarity for near-miss comparisons than analogical generalization).

vi. Using analogical generalization at testing time, a linear support vector machine can be layered on top of similarity-based retrieval for higher classification accuracy than comparable similarity-based or near-miss-based classifiers, while requiring nothing beyond prototypes to be stored in long-term memory (the same storage requirements as a comparable similarity-based classifier) and being more robust to the compression of those prototypes.

Experiment 4 tested the Analogical Boundary Learner (ABLe), an analogical twist on the SVM-KNN (Zhang et al. 2006), and a support-vector machine incarnation of the principle behind Halstead's analogical Bayes nets (Halstead 2011) and Liang & Forbus's SLogAn (Liang and Forbus 2015) – that analogical generalization can be used to build a feature space for a set of cases. The dimensions in this space correspond to assertions in the generalization, whose corresponding elements in each example are known once the example has been assimilated.

ABLe (which beings by selecting k similar examples as in SVM-KNN) was compared to a weighted k-nearest neighbor classifier with the same k, as well as the near-miss classifiers from Chapter 4. k=9 was used for a more direct comparison with the near-miss classifiers, which retrieved and tested at most 9 training examples at testing time before forcing a decision based on conflicting evidence. ABLe outperformed all these approaches for nearly all compression rates in all tasks. The improvement of ABLe over the less scalable near-miss classifier (Refined-near-misses), while consistently positive, was rarely significant, but ABLe used substantially less space (sub-linear long-term memory growth versus super-linear).

ABLe tended to perform better with a less compressed training case library, but it was nonetheless surprisingly resistant to the information loss. In Experiment 4, when classifying the Berlin25 dataset, as the assimilation threshold dropped (increasing compression), the weighted KNN approach saw its steepest decline in accuracy begin when the training case library had been compressed to about two thirds its original size, and reach an inflection point when the training library is compressed to about half. ABLe, on the other hand, saw stable performance in this phase on par with its uncompressed performance, and did not see a much steeper decline begin until the training library had been compressed to about a third of its original size. The other task in Experiment 4, while harder to visual interpolate and trust due to its smaller training set and

higher variance, lines up roughly with these same markers. This is all evidence for the fact that the analogical SVM approach can withstand substantially more compression than the similarity-based approach (or either near-miss approach, by extension).

7.2 Synthesis

The near-miss classifiers and ABLe both come up with hypotheses that are similar in spirit. In both, a positive-to-negative comparison with very few differences puts a boundary into focus, and that boundary extends out to other examples that are similar enough to be compared on the same dimensions. One big difference between the two is the rigidity of necessary conditions in the near-miss classifiers. A hypothesized criterion is discarded as soon as its probability drops below 1. This leaves it vulnerable to noise in the original input, the encoding, or the mapping

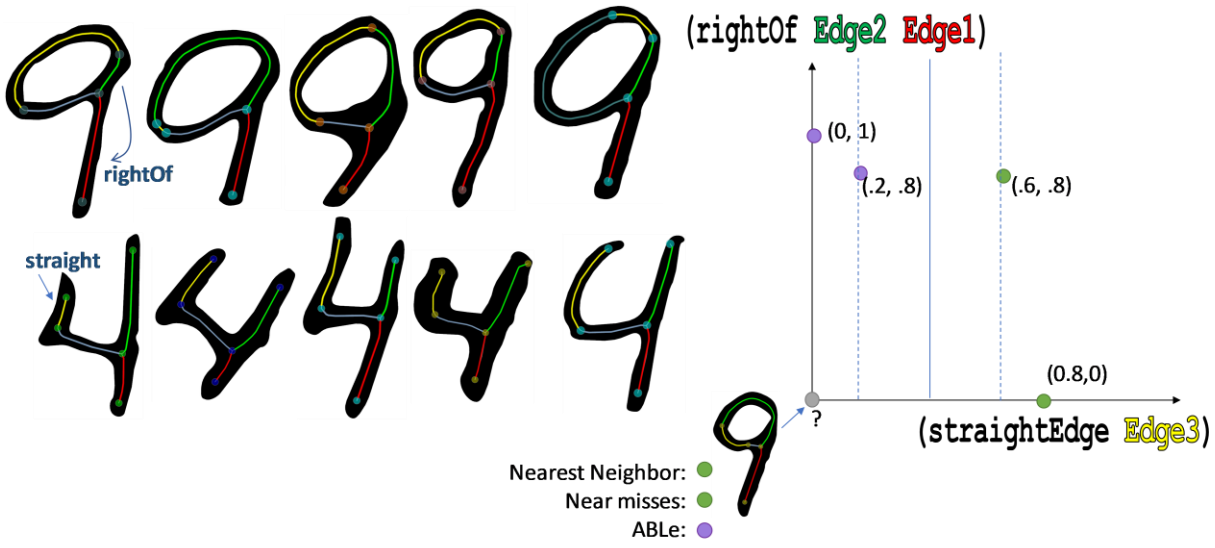


Figure 7.1: Fours and nines from MNIST, which, when generalized into two prototypes, reveal a critical dimension to ABLe that is unavailable to the near-miss learners because of the strict semantics of a necessary condition.

process. In a real-world scenario, we can expect few necessary conditions to survive in the long term under these conditions. We saw in our experiments that ABLe changes more smoothly even at the extremes (p=0 or p=1), making it more robust to this type of noise. ABLe does not saddle its hypotheses up with the semantics of necessary conditions. It instead learns margins that separate two classes (necessary and sufficient conditions in a binary decision). Figure 7.1 shows a situation where the margin framing may be more useful than the necessary condition framing. In this case, in our generalized prototype for *nine*, which assimilated the five examples in the top row, the yellow edge is almost never straight. In the fours in the bottom row, that edge is usually straight. Meanwhile, the relative positioning of the red and green edge, which has to do with the tilt of the digit, regresses in to the same mean for the two prototypes, revealing that all else being equal, the straightness of the yellow edge is the critical category boundary between these types of fours and these types of nines. The near-miss approaches with their rigid necessary conditions would not find this critical difference since it does not have probability 1.

In ABLe, we no longer need special handling for skolems like in near-miss learning, because the template generalization has assimilated all examples – from any category – whose comparisons will be relevant to learning the local boundary around the unlabeled example. Having assimilated the unlabeled example as well as relevant negative examples, the generalization will provide us with correspondences between their entities even if a corresponding entity does not exist in a similar positive example. Thus, we don't need to introduce a constrained variable in the hypothesis to transmit it through the positive and apply it to the unlabeled example.

The near-miss classifiers seemed to do poorly when too many hypotheses were generated. There may be an opportunity to use an ABLe-like approach at training time (or in rumination

between training and testing) to extract a select few hypothesized criteria for each prototype, which could then be applied at testing time like in the near-miss classifiers. In this case, the prototype would be the probe and seed for the generalization template. This would avoid the high testing-time computational cost of ABLe and may benefit from some of its strengths.

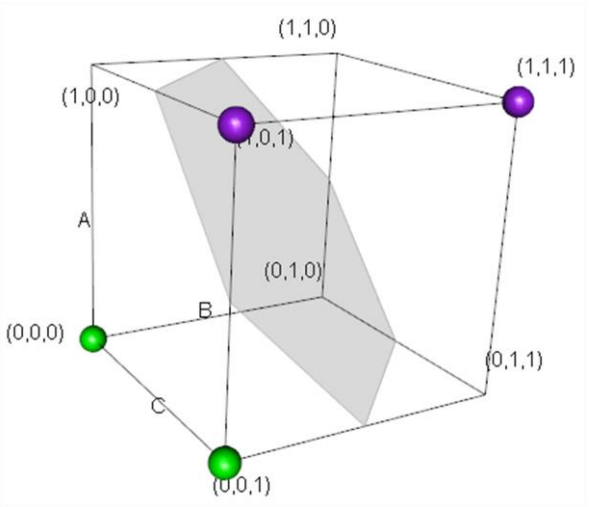


Figure 7.2: Near-misses can be misleading. Here the gray hyperplane is the true concept. The near-miss crosses the boundary along one of three possible dimensions to cross it on, because the true concept is edit distance between (0,0,0) and (1,1,1).

It is worth noting that for both ABLe and the near-miss classifiers, near-miss pairs can be misleading. A positive-negative pair could cross a boundary that is not orthogonal to the difference between them, even if it is the only difference. This could occur if the true concept is similarity-based or like the concept in Figure 7.2.

One elephant in the room with all of these results is that segmentation errors are both ubiquitous and high-impact, because of the rough and noisy nature of hand-drawn ink. As discussed at the end of Chapter 3, small topological errors in edge-segmentation can propagate to cause larger topological changes at higher levels of representation. Two crucial problems currently holding the present approach back are (1) how to qualitatively represent space with segmentation uncertainty, and (2) how to incorporate top-down signals into the segmentation process, for example, allowing the prospect of completing a large, salient-shaped edge-cycle to encourage the joining or two edges across a large gap in the ink.

7.3 Future Work

7.3.1 An entity-mapping network

As explained in Section 5.2, to represent an instance with a feature vector, ABLe requires that the instance be assimilated into a template generalization using SAGE. Every assertion in the template has a slot in the feature vector. Once an instance is assimilated into the template, SAGE's entity history is used to translate each template assertion to the instance so that it can be queried in the instance to assign a value to the slot. This is using entity correspondences as a shortcut to look for assertions in the instance that correspond to the assertions in the template, and it works because of the constraints of structure mapping (namely parallel connectivity and tiered identicality)

In fact, all that is needed to assign a feature vector to an instance is an injective mapping (one-to-one with leftovers allowed) between the entities in the template and the entities in the instance. This does not need to be achieved with SAGE, but using SAGE has some interesting properties. For one thing, it ensures that every assertion in the instance will be represented by a slot in the vector, since the template is guaranteed to contain an assertion that corresponds to it. It also provides one answer to the *what goes with nothing* problem (the matching of weak analogs, due to minor commonalities, that intuitively should have been leftovers; Section 3.3.2.1), because the template monotonically expands as instances are added to the feature space, providing a wider target for instances added thereafter to match against. Ideally this wider target will provide more options for sensible analogs. This is not necessarily a good answer to the issue, because this wider target means that weak analogs in each new instance are even less likely to end up unmatched (as leftovers). Moreover, if the encoding scheme is such that there is

virtually always superficial overlap, then the only way to ever expand the template is to assimilate a case with more entities than the template currently has. Other implications of using SAGE to find these entity-mappings were discussed in Section 5.2.1.2.

A major downside of using SAGE to build a template and assign feature vectors is that every instance to be assigned a vector requires at least one analogical match and a SAGE merge operation. This is expensive and prevents us from training large, or even moderately sized SVMs.

I propose a direction for future work in which the template can be analogically projected through a network of past analogies that have been distilled down to their numerical similarity score and entity correspondences (ideally including the support score for these correspondences, computed automatically by SME). Since these entity mappings are injective, a mapping composed by stringing them together along an acyclic path through the network is also injective. Similarity-based retrieval like MAC/FAC could be used to hook an unlabeled example into the entity mapping network. A template could be established by generalizing the unlabeled example with nearby examples, similar to ABLe, which generalizes with the retrieved examples. Alternatively, any other example in the network, including the unlabeled example, could serve as a template. Feature vectors could be assigned for examples further out in the network as long as there is a path connecting them to the template, because an entity mapping allows the template to be translated to each of them. In fact, alternative paths leading from the template to the example can produce alternative entity mappings, making it possible to assign multiple feature vectors for the same example. This is a form of analogical training set expansion in that the SVM could end up training on more instances than there are training examples.

Indirect entity mappings potentially make available orders of magnitude more data to train an SVM than ABLe without necessitating any additional testing-phase analogies or SAGE merge operations. All the analogical heavy lifting can precede the testing phase, allowing for classification time to depend only on the SVM training time – a change that would have improved ABLe's classification time by orders of magnitude. Each training-phase analogy could have a much greater return on investment than the analogies made in ABLe in its current form, since a link in the network can be reused by many different paths. Of course, the entity-mapping network concept is not dependent on SVMs – it could be used with any feature-based technique.

Note that the network could be developed offline by searching for new, productive links (analogies) between existing examples and eliminating counterproductive links. Some cases may be prismatic, acting as useful intermediate stops in understanding the match between two other cases. These cases may become a sort of hub in the entity-mapping network. Interestingly, an unlabeled example might be prismatic, and adding it to the network by comparing it to multiple cases in the network might improve the effectiveness of the network – a form of unsupervised learning. The clustering of instances (labeled or not) in a feature space – or multiple feature spaces corresponding to different templates in the network – could serve as a signal for the quality of the network for hill-climbing. The similarity scores of the links in the network provide an initial signal about the productivity/reliability of the links in the network. Moreover, the support scores that SME computes for individual entity correspondences provide an entity-specific signal about the quality of alternative entity mappings and may even be useful for combining the best entity correspondences from alternative paths as long as they do not break the injective requirement.

Figure 7.3 depicts a trained entity-mapping network, and an unlabeled example hooking into it via a few analogical matches. The lettered nodes represent training examples (letter denotes label) and prototypes (white). The solid links represent analogical comparisons made during training, like in the near-miss learner's training procedure, with link thickness representing numerical similarity score. The '?' node represents an unlabeled example and the dashed links represent comparisons made during testing to connect it to the network. Keeping around just the similarity scores the entity correspondences of the training comparisons, entity mappings can be inferred between any two indirectly connected nodes via a path between them, with alternative paths possibly contributing alternative entity

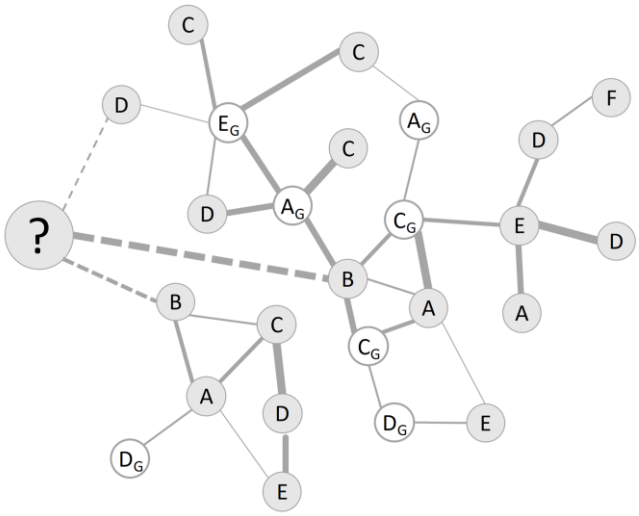


Figure 7.3: Hooking an unlabeled example up to a precomputed entity-mapping network in order to gain more training instances for an SVM at no additional cost in the testing phase. mappings for the same pair.

Finally, note that the indirect entity-mappings can help address the *what goes with nothing* problem (3.3.2.1). Consider Figure 7.4. Across the top are six examples from the TU Berlin dataset: (From left to right) one example of *Bathtub*, three of *Sailboat*, one of *Tea cup*, and one of *Cup*. All edge-cycle entities are marked in the examples with colored dashed lines. The circles in the column below each example also represent its edge-cycles, with colors to match. The line segments between circles in two neighboring columns represent the entity correspondences found in an analogical match between the examples in those columns (thickness

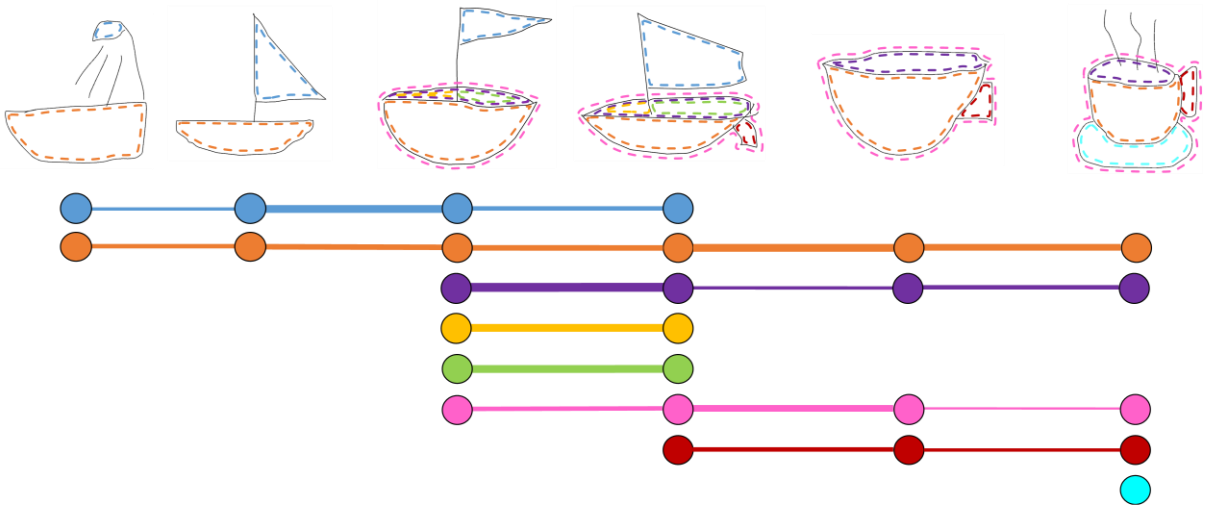


Figure 7.4: Mapping entities indirectly from a bathtub to a cup via a chain of direct entity mappings.

indicates the score of the entity correspondence). A direct mapping from the far-left example to the far-right example in Figure 7.4 would map both edge-cycles in the far left to edge-cycles in the far right. In the indirect mapping, there are entities left unmapped on both sides (e.g. the blue edge-cycle in the first row and the red edge-cycle in the seventh row) because they map onto different parts of the same intermediate example (e.g. fourth column). The benefits go beyond situations where the intermediate example has more entities. Notice how there are also entities left unmapped in the indirect mapping between the fourth column and the far-right column, this time because of an intermediate example that has fewer entities than both sides.

7.3.2 Feature Selection/Construction

By creating a dimension for every assertion in a generalization, whose representation is at least as large as its largest input case and at most as large as the sum of the sizes of its input cases, we create a very high-dimensional feature space in practice. The curse of dimensionality

is a term for the fraught nature of using a linear separator in high dimensional space, because the volume of the space grows so quickly with more dimensions that the training instances begin to look extremely sparse, even those that are intuitively relatively similar.

There's evidence to suggest that ABLe would do well to focus on fewer features in its hypotheses, either by feature selection or by using lasso (L1) regularization instead of ridge (L2) regularization. The work of Chen et al. (Chen et al. 2019) was significantly more successful on the MNIST dataset than the similarity-based approaches in this work, which are extremely similar in their classification procedures. One major difference is that the ones here do not use any feature selection. Another other major difference is that we filter cases down to 100 facts each, which is made more necessary by the more verbose descriptions sans feature selection. SLogAn (Liang and Forbus 2015) achieved classification performance that was competitive with the state of the art using lasso (L1) regularization to come up with hypotheses on a set of fewer dimensions. These hypotheses were also highly explainable thanks to their simplicity.

As alluded to in Section 5.2.1.1, the XOR problem suggests a role for feature construction, to capture higher order relationships between properties. Both feature selection and feature construction are made more promising by the fact that our features use ontologized predicates. An ontology may help to prioritize higher-order properties to add to the encoding, either with rules, or by applying patterns that have worked well in the past by analogy. Note that these forms of feature construction could be applied entirely at the knowledge level, allowing them to be applied to examples in long term memory, imagined examples (either by generalization or conceptual combination), or any other time the external representation (e.g. a sketch) is not available.

7.3.3 Sufficient conditions

The near-miss hypotheses from Chapter 4 are treated like hypotheses about necessary conditions, that they are pruned when they are found not to hold in all nearby (assimilated) positive examples, because necessary conditions should always be true of positive examples. Sufficient conditions should never be true of negative examples. In the structure mapping view, this functionally means they should never be true of *similar* negative examples, since logical inference only extends as far as cases similar enough to draw a reliable mapping. Sufficient conditions could be hypothesized from near-miss comparisons just like necessary conditions. Instead of being subject to pruning whenever positive examples are generalized, sufficient condition hypotheses would be subject to pruning whenever near-misses are detected. If a sufficient condition hypothesis for one case holds at all ($p > 0$) in a near-miss case, it must not be sufficient for membership and can be pruned.

There are additional complications to address. For one thing, new sufficient condition hypotheses would need to be retroactively pruned by old near-misses, or else noisy sufficient condition hypotheses coming from the most recent near-misses would drown out the signal of the older, battle-tested hypotheses. This is not a concern for necessary condition hypotheses since the relevant historical information is evident from the prototype.

Sufficient conditions learned in this way are intriguing, because they would be both born of, and filtered by, near-misses. Generalization would only affect them in so far as it could add or subtract near-misses for a prototype. They are also intriguing more generally because at testing time, they tell us about what things are, rather than what they aren't. (The near-miss learners

classify by assuming that something similar is of the same label unless given a reason it is not, but this is more of an inferential leap than confirming that a sufficient condition holds.)

7.3.4 SAGE WM

Classifying an example with ABLe includes three potentially expensive subroutines: (a) analogical matches, of which there are 2K-1 for k retrieved training examples, and the last k of which are increasingly expensive as the template generalization grows, (b) SAGE's merge step, which is executed k times and becomes increasingly expensive as the template generalization grows, and (c) training the SVM. Assuming the cases contain n facts each, then as long as $k < n$, which was the case in the experiments here, the relative theoretical worst-case time complexity of the steps above are ordered (a) > (c) > (b). However, in the experiments here, we consistently found that in terms of the real time spent, (b) > (a) > (c). In fact, for a given testing round, the average time spent per merge step was 15 to 100 times the average time spent per analogical match.

The complexity of SAGE's merge step is theoretically fast. It is linear to the number of assertions in the input cases. It was the bottleneck for an engineering reason – knowledge base transactions. SAGE effectively deletes and rewrites the entire generalization on each successive merge, and the KB transactions piled up.

Since it is ad hoc, there is no reason besides convenience of implementation that the template generalization should be written to the knowledge base (long term memory). This is a perfect job for SAGE WM (Kandaswamy 2017), which uses the SAGE algorithm to build ephemeral generalizations in working memory. Any future work would be wise to implement the ABLe algorithm using SAGE WM.

7.3.5 Independent near-miss threshold

At their introduction, near-misses were intended to be near. The setting of a fixed near-miss threshold (usually 0.7-0.9) seemed somewhat arbitrary and added a parameter (McLure et al. 2010). In this work, we tied it to the assimilation threshold for the elegance of uniting positive↔positive comparisons (for generalization) and positive↔negative comparisons (near-misses) under the same trigger – a universal similarity threshold that parameterized the propensity to compare. This was likely a limitation of the approaches in Chapter 4. Near-misses should be near. For a far-miss, which were being produced in the lower half of the similarity threshold spectrum in Experiment 3, the number of potential explanations in terms of differing properties was explosive, to say nothing of the unreliability of a far-miss mapping.

Future work using near-miss hypotheses should explore decoupling the thresholds and use smarter conditions under which to detect and harvest criteria from a near-miss. Perhaps the most directly relevant measure of a near-miss's focus and value is its number of candidate inferences (and reverse candidate inferences). These are what determine how many alternative explanations will be hypothesized from a comparison. Just as we amplify the weight of hypotheses corresponding to larger generalizations, we could amply those that were one of fewer explanations for a near-miss.

7.3.6 Support Vectors and Valuable Examples

Multiple experiments in this work bore evidence for the claims that increasing compression eventually degrades performance, and in general more compression leads to more error. This was true of the weighted k-nearest-neighbors classifier (Retrieve-WKNN), the Prototype-near-misses classifier, and ABLE. The Refined-near-misses classifier was improved by some

compression in its prototypes, which it kept in addition to, rather than in lieu of, the individual training examples that were assimilated into them. This was ostensibly because the compressed prototypes provided a benefit for locally refining criteria, while not harming retrieval during near-miss discovery and classification by replacing individual examples. Some individual assimilated examples may reveal a more focused view of a (local) boundary by lying near it, and some may provide informative indices into prototypes and learned theories by matching unlabeled examples very cleanly.

Keeping around all training examples is intractable. SAGE's method of compression already preserves individual examples that are sufficiently unique, but there is evidently value in also keeping around some of the individual examples that were assimilated. One family of techniques worth exploring are those that selectively retain/discard assimilated examples. In the author's view this is cognitively plausible (I simultaneously recall the details of my first mountain bike and the details of a friend's particularly salient mountain bike, while seeing how both fit comfortably into my model of a typical mountain bike). More to the point, it provides an opportunity to retain more of the benefit of individual examples while achieving sub-linear long-term memory demands and benefitting from prototypes for hypothesis refinement, rerepresentation/encoding scheme adjustment, forming new sub-concepts by unsupervised learning, and finding other ontological or mereological links.

The ABLE technique provides an interesting signal for example valuation – the support vectors. These correspond to the examples that determine the margin between concepts and may be a good signal for which assimilated examples to retain vs. discard.

7.3.7 Using one-vs-all classifier in the n-ary ensemble

The one-vs-all classifier is particularly interesting avenue of future work since it is more similar to the semantics of near-miss criteria hypotheses. To the author's knowledge, both extend more naturally than a one-vs-one model to training and testing examples with an arbitrary number of labels (non-mutually exclusive categories), which are likely to be common in learning machines that have rich ontologies. Even in the mutually exclusive case, the one-vs-one classifier seems less explainable, since the ultimate label, chosen from a set of *n*, is a result of $\frac{n(n-1)}{2}$ real-valued votes, all voting on the basis of different pairwise comparisons (e.g. *apple* vs. *pear*, *pear* vs. *house*, *apple* vs. *house*,…). In the one-vs-all classifier for mutually exclusive concepts, there are only *n* real-valued votes. These votes are about whether something is or isn't in a category, which seems a more ubiquitous type of step in explaining reasoning than a forced choice between two categories, which may be quite different (e.g. *apple* vs. *house*).

7.3.8 Resource awareness & open parameters

Some parameters in this family of approaches are left open because they are seen as useful knobs for a higher-level control process. But a theme throughout this thesis has been a concern for understanding and controlling tractability tradeoffs. For example, under a time constraint, we know that we can afford to retrieve more cases (a higher k value) as the similarity threshold rises or as the fact limit drops. Our digital assistants should know the trade-offs in their own reasoning processes, and analogical learning is no exception. These trade-offs should be learned. A more sophisticated learned model of a trade-off (qualitative and qualitative), alongside probes into resource usage of subprocesses, would not only allow for precise controls on resource expenditure, but could even facilitate the discovery of clever optimizations.

7.3.9 Spatial representations with uncertainty

The binary qualitative representations used here do not take full advantage of the support vector machine's hypothesis. Qualitative representations derived from a sketch are derived using thresholds, but when measurements are near the threshold, we may have less confidence in the attribute or relation. Uncertainty in a dimension that is highly discriminative will defer the classification boundary to other dimensions. Figure 7.5 shows a three-assertion feature space in which the separator has come up with a single discriminating feature (A) for cases with binary features, which land at the vertices of the unit cube. However, if the features in the unlabeled example can have continuous values – or confidence values – the decision is a linear function of their confidence values. Hence, there is a range of confidences for A in which the decision is left to the confidence in B and C.

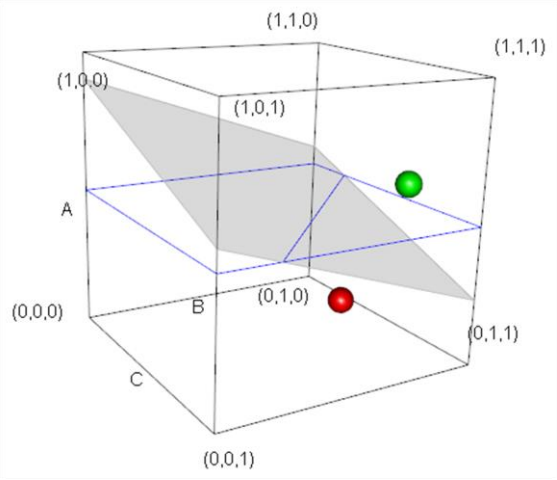


Figure 7.5: ABLe would naturally extend to spatial representations with uncertainty. Here, the SVM's logical hypothesis is just A when assertions (features) are binary. When A is uncertain with confidence=0.5, the SVM's hypothesis is B ^ C.

7.3.10 Sketch generation

Sketching is a powerful learning medium for humans, in part, because any imagined concept can be generated contemporaneously. Compare that to a photograph, for example. Often one seeks to generate a very particular instance, either to better learn the category or to investigate the viability of the instance itself (this could be biological viability for someone trying to

document and taxonomize new deep-sea species, or it could be spatial viability for an urban planner trying to route a bike path).

As discussed at the beginning of Chapter 4, the learning approaches here were designed with active learning by sketch generation as a primary objective. The learned representations consist of structured representations of prototypes in which the relationships and attributes are probabilistic and domain-general. The hypotheses are embedded in the representations and are expressed in terms these relationships and attributes. One promising approach for testing a conceptual boundary by query synthesis would be to modify a prototype in such a way that crosses a hypothesized boundary to produce a near-miss, like in the Automatic Near Misses Generation technique (Gurevich, Markovitch, and Rivlin 2006), and then instantiate the resulting representation using the probabilities in the prototype to give weight to competing characteristics. Similarly, in ABLe, we might attempt to test an exact point in the feature space (e.g. on the separating hyperplane) by using weights on qualitative constraints. To these ends, we finish by describing sketch generation using simulations of spring systems.

7.3.10.1 Spring systems

Spatial constraint satisfaction via spring systems has been used for automatic graph drawing, where it has been demonstrated to be capable of enforcing various spatial relationships such as alignment, even spacing, and symmetry (Ryall, Marks, and Shieber 1997). There are two types of springs in our simulator. *Helical springs* apply a force along a straight line toward an equilibrium length. The springs in jack-in-the-boxes and a click-pens are helical springs. *Torsion springs* add angular force toward an equilibrium angle. The spring in a mouse trap is a torsion spring.

While not ideal for global constraint optimization (Hower & Graf (1996) provide a survey), spring systems have a few very appealing properties for sketch generation from the structured qualitative spatial descriptions used here. First, spring systems fail gracefully and quickly. This work is aimed toward applying sketch generation to novel internal representations, i.e. imagined examples. These internal representations are relational and qualitative, but also may have probabilities or other types of weights associated with them, like in a prototype found by SAGE. They also may be over-constrained. Spring systems deal very gracefully with over-constrained systems. They settle into a local energy minimum (the energy stored in the springs). This means conflicting subsystems of springs will settle into some sort of compromise position, which may or may not be intuitive or globally optimal. They tend to do so quite quickly depending on the parameters of the physical simulation. Figure 7.6 shows set of 3 nodes with springs added to satisfy qualitative constraints, which eventually collectively describe an impossible (over-constrained) system. From left to right, we see the same three-node system (1) with two unconstrained helical springs, (2) with a *same-length* constraint added to the two springs, (3) with a *right-angle* torsion spring added between a pair of edges, (4) with a third helical spring added that is constrained to have the same length as the first two, (5) with a second right-angle

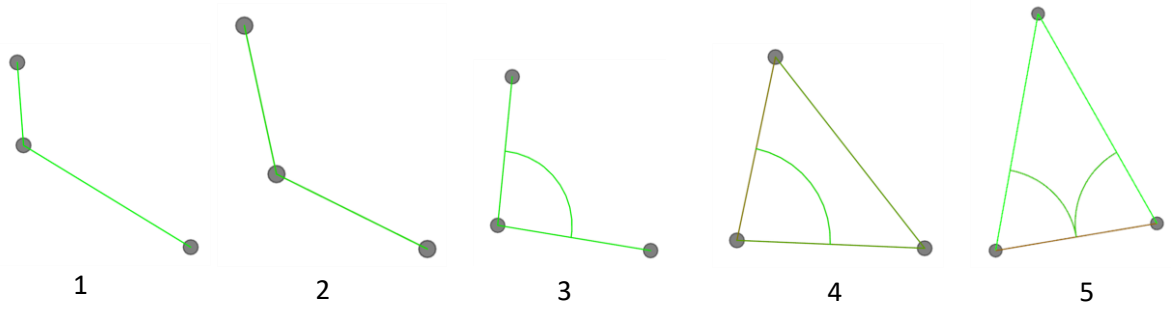


Figure 7.6: The low-energy states found by our (implemented) spring-system simulator for an increasingly constrained spring system.

torsion spring added. The colors of springs reflect the energy stored in them (how far from satisfied they are). In the last two configurations, the system is over-constrained, because it is spatially impossible to satisfy all the qualitative relationships at once. In (4), the triangle has found a compromise between a right triangle and an equilateral triangle. In (5), the right-angle constraints have driven the opposite vertex away, until the same length constraints push back to equilibrium, settling on an isosceles triangle. Note that adjusting the spring constant *k* for some of the springs would change the balance found between the conflicting constraints.

Second, spring systems are appealing because the core mechanism is extremely simple and intuitive, and its generalization to new spatial relationships and new levels of granularity hinges on the ability to come up with appropriate spring metaphors. Edge length and angles have obvious implementations with helical and torsion springs, respectively, as in Figure 7.6. Our qualitative descriptions capture configurations of approximately constant-curvature edges. We allow generated edges to have any constant curvature by implementing them as circle arcs. We introduce a new control point for each arc halfway along its arc length. This creates an inscribed angle with the endpoints of the arc that is a right angle when the arc is a semicircle (Figure 7.7, left). Thus, we can reuse the angle spring constraints to implement curvature, arc length, and convexity (with respect to an edge-cycle). Torsion springs were also used to implement parallel relationships between straight lines and orientation attributes like *horizontal* or *axis-aligned*. Helical springs implemented positional relationships and *same-length*. Figure 7.7 (right) shows a generated sketch of a puzzle-piece that was reconstructed from an encoded edge-level structural description using this library of spring-based metaphors. The edges (circle arcs) are in violet, the junctions between them are large dark blue circles, the control points are the smaller gray circles, and the springs are overlaid in green.

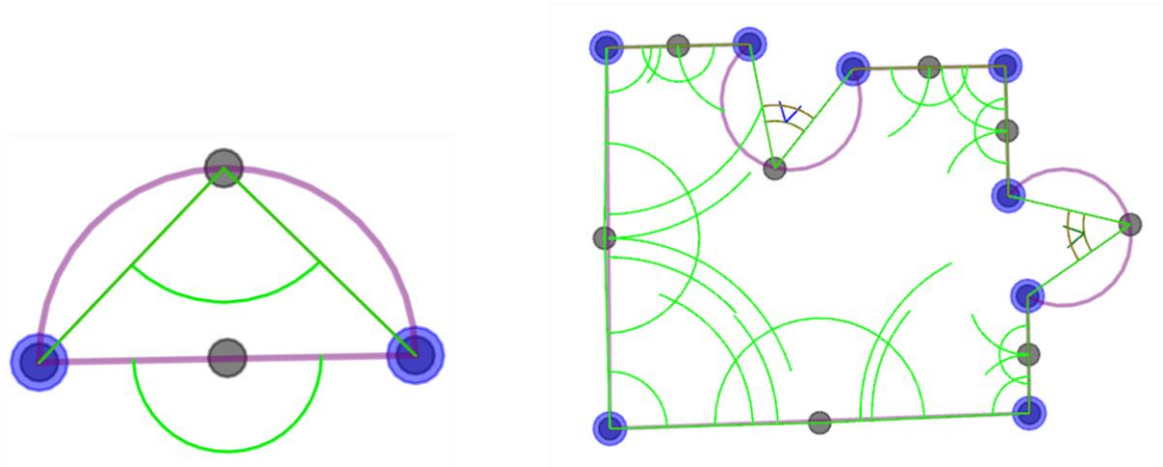


Figure 7.7: (Left) Two constant-curvature arcs (purple), and torsion springs for enforcing their qualitative curvature (straight vs semicircle) via an inscribed control point (gray). (Right) A spring system implementing a qualitative description of a jigsaw puzzle piece.

The implementations of *parallel* and *same-length* depart from a simulation of normal physical springs. Instead of having a fixed resting position (a length for a helical spring or an angle for a torsion spring), the springs used to implement these relationships have dynamic resting positions based on the state of other elements in the system. Springs with dynamic equilibria can potentially support constraints for representations of edge-cycles and more abstract objects, because the forces on individual nodes (vertices or edge control points) can be functions of properties of more abstract elements, e.g. the center of mass or major axis of an edge-cycle. These more abstract properties are a crucial avenue of future work because of the classification value of sparser representations like edge-cycles. To date only a limited number of edge-cycle relationships have been implemented, e.g. positional relationships. On the downside, springs with dynamic equilibria might compromise the first advantage of spring systems that we discussed – the ability to find a local minimum quickly and gracefully.

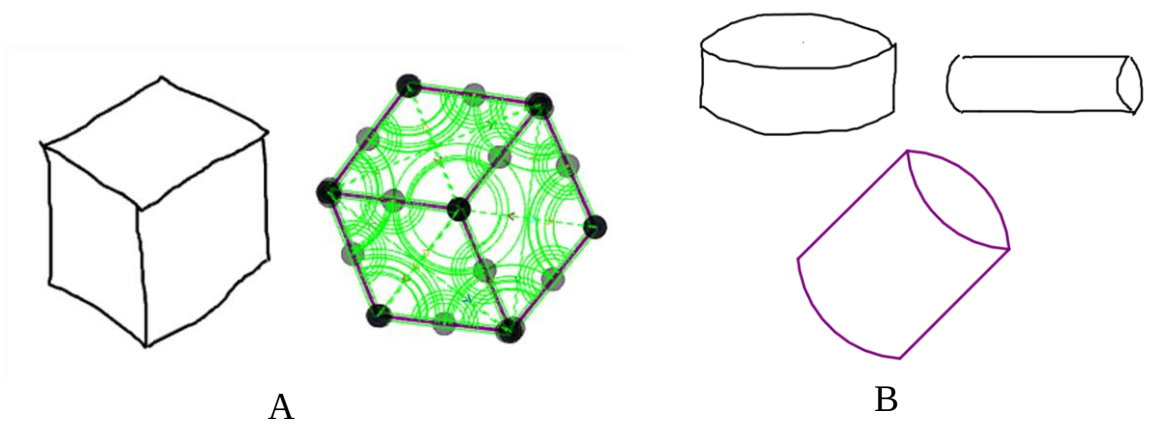


Figure 7.8: (A) The hand-drawn cube was encoded using the unfiltered hybrid edge-cycle/edge scheme from Chapter 3. Spring systems were used draw a new cube, completing a representational loop. All springs are overlaid on the new cube in green. (B) The two hand-drawn cylinders (black ink; top) were encoded and then generalized using SAGE, and the SAGE generalization was drawn using spring systems (purple ink; bottom).

The third appeal of spring systems is that they easily handle a mixture of hard and soft constraints because they are all implemented quantitatively in the spring constant k. Numerical weights, such as probabilities in a SAGE generalization (e.g. Figure 7.8, bottom), can be applied using spring constants.

The final appeal is pragmatic – spring systems have a familiar, intuitive visual rendering which may make them easier to develop and debug in a sketching application.

As the spring systems get larger and more complex, there come to be more and more local energy minima to fall into. We have had some success exploring multiple local minima using random perturbations to elements in the system, but there may be a role for unsupervised, non-linear methods like deep learning in learning how to best perturb a given configuration to achieve a lower energy state.

Note that this approach can potentially extend to novel conceptual combination by analogy (e.g. a wheelbarrow of pumpkins or a truck towing a boat), since its internal representations are composable.

8 References

Armstrong, SL, LR Gleitman, and H. Gleitman. 1983. “What Some Concepts Might Not Be.” *Cognition*.

Aubert, M. et al. 2018. “Palaeolithic Cave Art in Borneo.” *Nature* 564(7735):254–57.

Bai, Xiang, Longin Jan Latecki, and Wen-Yu Liu. 2006. “Skeleton Pruning by Contour Partitioning.” *Proc. 13th Int. Conf. Discret. Geom. Comput. Imag.* 567–79.

Bangia, Sachet et al. 2017. “Redistricting: Drawing the Line.” *ArXiv e-prints* 1–44.

Barbella, David and Kenneth D. Forbus. 2013. “Analogical Word Sense Disambiguation.” *Advances in Cognitive Systems* 2:297–315.

Blass, Joseph A. and Kenneth D. Forbus. 2016. “Modeling Commonsense Reasoning via Analogical Chaining: A Preliminary Report.” Pp. 556–61 in *Proceedings of the 38th Annual Meeting of the Cognitive Science Society*. Philadelphia, PA.

Bloch, Isabelle, Olivier Colliot, and Roberto M. Cesar. 2006. “On the Ternary Spatial Relation ‘between’ RID B-2092-2012 RID C-4120-2012.” *Ieee Transactions on Systems Man and Cybernetics Part B-Cybernetics* 36(2):312–27.

Bloodgood, Michael. 2018. “Support Vector Machine Active Learning Algorithms with Query-by-Committee Versus Closest-to-Hyperplane Selection.” *Proceedings - 12th IEEE International Conference on Semantic Computing, ICSC 2018* 2018–Janua(January):148–55.

Blum, H. 1967. “A Transformation for Extracting New Descriptors of Shape.” Pp. 362–80 in *Proceedings of the Symposium on Models for the Perception of Speech and Visual Form*, edited by W. W. Dunn. MIT Press, Cambridge, Mass.

Buckley, S. 1979. *Sun Up to Sun Down.* McGraw Hill: New York.

Chang, Chih-Chung and Chih-Jen Lin. 2011. “Libsvm.” *ACM Transactions on Intelligent Systems and Technology* 2(3):1–27.

Chang, Maria D. and Kenneth D. Forbus. 2014. “Using Analogy to Cluster Hand-Drawn Sketches for Sketch-Based Educational Software.” *AI Magazine* 35(1):76.

Chang, Maria de los Angele. 2016. “Capturing Qualitative Science Knowledge with Multimodal Instructional Analogies.” Northwestern University.

Chen, Kezhen, Irina Rabkina, Matthew D. Mclure, and Kenneth D. Forbus. 2019. “Human-like Sketch Object Recognition via Analogical Learning.” in *Proceedings of the 33rd Conference on Artificial Intelligence (AAAI 2019)*.

Cleuziou, G., L. Martin, and C. Vrain. 2003. “Disjunctive Learning with a Soft-Clustering Method.” *Inductive Logic Programming*.

Cumby, Chad M. and Dan Roth. 2003. “Feature Extraction Languages for Propositionalized Relational Learning.” *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Workshop on Learning Statistical Models from Relational Data* 24–31.

Dehghani, M. and AM Lovett. 2006. “Efficient Genre Classification Using Qualitative Representations.” *ISMIR*.

Dehghani, Morteza. 2009. “A Cognitive Model of Recognition-Based Moral Decision Making.” Northwestern University.

Duchin, Moon. 2018. “Outlier Analysis for Pennsylvania Congressional Redistricting.” 1–17.

Eitz, Mathias, James Hays, and Marc Alexa. 2012. “How Do Humans Sketch Objects?” *ACM Transactions on Graphics* 31(4):1–10.

Falkenhainer, Brian, Kenneth D. Forbus, and Dedre Gentner. 1989. “The Structure-Mapping Engine: Algorithm and Examples.” *Artificial Intelligence* 41(1):1–63.

Finlayson, MA and PH Winston. 2006. “Analogical Retrieval via Intermediate Features: The Goldilocks Hypothesis.”

Firestone, Chaz and Brian J. Scholl. 2014. “‘Please Tap the Shape, Anywhere You Like’: Shape Skeletons in Human Vision Revealed by an Exceedingly Simple Measure.” *Psychological Science* 25(2):377–86.

Forbus, Kenneth D. et al. 2018. “Sketch Worksheets in STEM Classrooms: Two Deployments.” in *Proceedings of IAAI 2018*.

Forbus, Kenneth D., Maria Chang, Matthew McLure, and Madeline Usher. 2017. “The Cognitive Science of Sketch Worksheets.” *Topics in Cognitive Science* 9(4):921–42.

Forbus, Kenneth D., Ronald W. Ferguson, Andrew Lovett, and Dedre Gentner. 2016. “Extending SME to Handle Large-Scale Cognitive Modeling.” *Cognitive Science*.

Forbus, Kenneth D., Dedre Gentner, and Keith Law. 1995. “MAC/FAC: A Model of Similarity-Based Retrieval.” *Cognitive Science* 19(2):141–205.

Forbus, Kenneth D., Matthew Klenk, and Thomas Hinrichs. 2009. “Companion Cognitive Systems: Design Goals and Lessons Learned So Far.” *IEEE Intelligent Systems* 24(4):36–46.

Friedman, Scott E. 2012. “Computational Conceptual Change: An Explanation-Based

Approach." Northwestern University.

Friedman, Scott E. 2015. "Exploiting Graph Structure to Summarize and Compress Relational Knowledge." *Proceedings of the 28th International Workshop on Qualitative Reasoning*.

Gaddam, Shekhar, Vir Phoha, and Kiran Balagani. 2007. "K-Means+ID3: A Novel Method for Supervised Anomaly Detection by Cascading K-Means Clustering and ID3 Decision Tree Learning Methods." *IEEE Transactions on Knowledge and Data Engineering* 19(3):345–54.

Gentner, D. 1983. "Structure-Mapping: A Theoretical Framework for Analogy." *Cognitive science*.

Goldschmidt, Gabriela. 1991. "The Dialectics of Sketching." *Creativity Research Journal* 4(2):123–43.

Gross, Mark D. 1996. "The Electronic Cocktail Napkin—a Computational Environment for Working with Design Diagrams." *Design Studies* 17(1):53–69.

Gurevich, N., S. Markovitch, and E. Rivlin. 2006. "Active Learning with near Misses." *Proceedings of the National Conference on Artificial Intelligence.* Vol. 21.(1).

Ha, David and Douglas Eck. 2017. "A Neural Representation of Sketch Drawings."

Halstead, DT. 2011. "Statistical Relational Learning through Structural Analogy and Probabilistic Generalization."

Hammond, T. and R. Davis. 2004. "Automatically Transforming Symbolic Shape Descriptions for Use in Sketch Recognition." *AAAI*.

Hammond, T. and R. Davis. 2006. "LADDER: A Language to Describe Drawing, Display, and Editing in Sketch Recognition." *ACM SIGGRAPH 2006 Courses*.

Hinrichs, TR and KD Forbus. 2012. "Learning Qualitative Models by Demonstration." *AAAI*.

Hoffman, D. D. and W. A. Richards. 1984. "Hoffman (1984) Parts of Recognition.Pdf." *Cognition* 18(1–3):65–96.

Hower, Walter and Winfried H. Graf. 1996. "A Bibliographical Survey of Constraint-Based Approaches to CAD, Graphics, Layout, Visualization, and Related Topics." *Knowledge-Based Systems* 9(7):449–64.

Kandaswamy, Balasubramanian. 2017. "Comparison Driven Representational Change." *Dissertation Abstracts International: Section B: The Sciences and Engineering* 78(5–B(E)).

Klenk, Matthew and Ken Forbus. 2009. "Analogical Model Formulation for Transfer Learning in

AP Physics." *Artificial Intelligence* 173(18):1615–38.

Klenk, Matthew, Ken Forbus, Emmett Tomai, and Hyeonkyeong Kim. 2011. "Using Analogical Model Formulation with Sketches to Solve Bennett Mechanical Comprehension Test Problems." *Journal of Experimental and Theoretical Artificial Intelligence* 23(3):299–327.

Kok, Stanley and Pedro Domingos. 2007. "Statistical Predicate Invention." *Proceedings of the 24th international conference on Machine learning - ICML '07* 433–40.

Kuehne, S., K. Forbus, D. Gentner, and B. Quinn. 2000. "SEQL: Category Learning as Progressive Abstraction Using Structure Mapping." *Proceedings of the 22nd Annual Meeting of the Cognitive Science Society*.

Lake, Brenden M., Rusian Salakhutdinov, and Joshua B. Tenenbaum. 2015. "Human-Level Concept Learning through Probabilistic Program Induction." *Science* 350(6266):1332–38.

Landwehr, Niels, Andrea Passerini, Luc De Raedt, and Paolo Frasconi. 2010. "Fast Learning of Relational Kernels." *Machine Learning* 78(3):305–42.

LeCun, Y., Y. Bottou, Y. Bengio, and P. Haffner. 1998. "Gradient-Based Learning Applied to Document Recognition." P. 86:2278–2324 in *Proceedings of the IEEE*.

Lee, WS, L. Burak Kara, and TF Stahovich. 2007. "An Efficient Graph-Based Recognizer for Hand-Drawn Symbols." *Computers & Graphics*.

Liang, Chen and Kenneth D. Forbus. 2015. "Learning Plausible Inferences from Semantic Web Knowledge by Combining Analogical Generalization with Structured Logistic Regression." *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence (AAAI)* (January):551–57.

Lladós, J., E. Martí, and JJ Villanueva. 2001. "Symbol Recognition by Error-Tolerant Subgraph Matching between Region Adjacency Graphs." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 23(10):1137–43.

Lockwood, K., A. Lovett, and K. Forbus. 2008. "Automatic Classification of Containment and Support Spatial Relations in English and Dutch." *Spatial Cognition VI. Learning, Reasoning, and Talking about Space.* 283–94.

Lovett, A. and K. Forbus. 2011. "Organizing and Representing Space for Visual Problem-Solving." *Proceedings of 25th Qualitative Reasoning Workshop*.

Lovett, A., S. Kandaswamy, M. McLure, and K. Forbus. 2012. "Evaluating Qualitative Models of Shape Representation." *Proceedings of the 26th International Workshop on Qualitative Reasoning*.

Lovett, A., E. Tomai, K. Forbus, and J. Usher. 2009. "Solving Geometric Analogy Problems

Through Two-Stage Analogical Mapping." *Cognitive science*.

Lovett, Andrew. 2012. "Spatial Routines for Sketches: A Framework for Modeling Spatial Problem-Solving." Northwestern University.

Lovett, Andrew, M. Dehghani, and Kenneth Forbus. 2006. "Efficient Learning of Qualitative Descriptions for Sketch Recognition." *Proceedings of the 20th International Qualitative Reasoning Workshop* (1994).

Lovett, Andrew, Morteza Dehghani, and Kenneth Forbus. 2007. "Incremental Learning of Perceptual Categories for Open-Domain Sketch Recognition."

Lovett, Andrew, Morteza Dehgnani, and Kenneth Forbus. 2007. "Constructing Spatial Representations of Variable Detail for Sketch Recognition." *AAAI Spring Symposium on Control Mechanisms for Spatial Knowledge Processing in Cognitive / Intelligent Systems*.

Lovett, Andrew and Kenneth Forbus. 2011. "Cultural Commonalities and Differences in Spatial Problem-Solving: A Computational Analysis." *Cognition* 121(2):281–87.

Lovett, Andrew and Kenneth Forbus. 2017. "Modeling Visual Problem Solving as Analogical Reasoning." *Psychological Review* 124(1):60–90.

Lovett, Andrew and Holger Schultheis. 2014. "Modeling Spatial Abstraction during Mental Rotation." *Proceedings of the Annual Meeting of the Cognitive Science Society* 36(36):930–35.

Lovett, Andrew, Emmett Tomai, Kenneth Forbus, and Jeffrey Usher. 2009. "Solving Geometric Analogy Problems through Two-Stage Analogical Mapping." *Cognitive science* 33(7):1192–1231.

Lowet, Adam S., Chaz Firestone, and Brian J. Scholl. 2018. "Seeing Structure: Shape Skeletons Modulate Perceived Similarity." *Attention, Perception, and Psychophysics* 80(5):1278–89.

Machery, E. 2009. "Doing without Concepts."

McFate, C., KD Forbus, and TR Hinrichs. 2013. "Using Narrative Function to Extract Qualitative Information from Natural Language Texts: A Preliminary Report." *qrg.northwestern.edu*.

McLure, M., S. Friedman, and K. Forbus. 2010. "Learning Concepts from Sketches via Analogical Generalization and Near-Misses." *Proceedings of the 32nd Annual Conference of the Cognitive Science Society (CogSci)*.

Mclure, Matthew D., Scott E. Friedman, and Kenneth D. Forbus. 2015. "Extending Analogical Generalization with Near-Misses." *Proceedings of the 29th Conference on Artificial Intelligence (AAAI 2015)* (1):565–71.

McLure, MD and KD Forbus. 2012. “Encoding Strategies for Learning Geographical Concepts via Analogy.” *qrg.northwestern.edu*.

McLure, MD, SE Friedman, and KD Forbus. 2011. “Edge-Cycles: A Qualitative Sketch Representation to Support Recognition.” *Proceedings of the 25th International Workshop on Qualitative Reasoning*.

McLure, MD, S. Kandaswamy, and KD Forbus. 2015. “Finding Textures in Sketches Using Planar Ising Models.” in *Proceedings of the 28th International Workshop on Qualitative Reasoning*. Minneapolis, MN.

Michalski, RS. 1983. “A Theory and Methodology of Inductive Learning.” *Artificial intelligence*.

Michalski, Ryszard S., Jaime G. Carbonell, and Tom M. Mitchell, eds. 1983. *Machine Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg.

Mokhtarian, Farzin and Riku Suomela. 1998. “Robust Image Corner Detection through Curvature Scale Space.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(12):1376–81.

Muggleton, S. 1995. “Inverse Entailment and Progol.” *New generation computing*.

Muggleton, S. and C. Feng. 1992. “Efficient Induction of Logic Programs.” *Inductive logic programming*.

Muggleton, S. H., H. Lodhi, A. Amini, and M. J. E. Sternberg. 2006. “Support Vector Inductive Logic Programming.” *Studies in Fuzziness and Soft Computing* 194:113–35.

Navia-Vázquez, A. and E. Parrado-Hernández. 2006. “Support Vector Machine Interpretation.” *Neurocomputing* 69(13–15):1754–59.

Osherson, DN and EE Smith. 1981. “On the Adequacy of Prototype Theory as a Theory of Concepts.” *Cognition*.

Ouyang, Tom Y. and Randall Davis. 2011. “ChemInk.” P. 267 in *Proceedings of the 15th international conference on Intelligent user interfaces - IUI ’11*. New York, New York, USA: ACM Press.

Peura, Markus and Jukka Iivarinen. 1997. “Efficiency of Simple Shape Descriptors.” Pp. 443–451 in *Proceedings of the Third International Workshop on Visual Form*.

Plotkin, GD. 1970. “A Note on Inductive Generalization.” *Machine intelligence*.

Purcell, AT and JS Gero. 1998. “Drawings and the Design Process: A Review of Protocol Studies in Design and Other Disciplines and Related Research in Cognitive Psychology.”

Design Studies 9(4):389–430.

Quillin, Kim and Stephen Thomas. 2015. “Drawing-to-Learn : A Framework for Using Drawings to Promote Model-Based Reasoning in Biology.” *CBE life sciences education* 14(1):1–16.

Quinlan, JR. 1990. “Learning Logical Definitions from Relations.” *Machine learning*.

Quinlan, JR. 1991. “Determinate Literals in Inductive Logic Programming.” *IJCAI*.

De Raedt, Luc, Niels Landwehr, Andrea Passerini, and Paolo Frasconi. 2006. “KFOIL: Learning Simple Relational Kernels.” *Proceedings of the 21st national conference on Artificial intelligence (AAAI ’06)* 389–94.

Richardson, M. and P. Domingos. 2006. “Markov Logic Networks.” *Machine Learning* 62(1–2):107–36.

Ryall, K., J. Marks, and S. Shieber. 1997. “An Interactive Constraint-Based System for Drawing Graphs.” *Proceedings of the 10th annual ACM symposium on User interface software and technology*.

Sagi, Eyal, Dedre Gentner, and Andrew Lovett. 2012. “What Difference Reveals about Similarity.” *Cognitive Science* 36(6):1019–50.

Saund, Eric. 2003. “Finding Perceptually Closed Paths in Sketches and Drawings.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25(4):475–91.

Saund, Eric, James Mahoney, David Fleet, Dan Larner, and Edward Lank. 2002. “Perceptual Organization as a Foundation for Intelligent Sketch Editing.” Pp. 118–25 in *AAAI Spring Symposium on Sketch Understanding*.

Schneider, Rosália G. and Tinne Tuytelaars. 2014. “Sketch Classification and Classification-Driven Analysis Using Fisher Vectors.” *ACM Transactions on Graphics* 33(6):1–9.

Schohn, Greg and D. Cohn. 2000. “Less Is More: Active Learning with Support Vector Machines.” *Machine Learning-International Workshop Then Conference*- 839–846.

Schwartz, B. and D. Reisberg. 1991. “Learning and Memory.”

Siddiqi, K. and S. Pizer. 2008. *Medial Representations: Mathematics, Algorithms and Applications*. Vol. 37. Springer Science & Business Media.

Siddiqi, Kaleem, Ali Shokoufandeh, Sven J. Dickinson, and Steven W. Zucker. 1999. “Shock Graphs and Shape Matching.” *International Journal of Computer Vision* 35(1):13–32.

Smith, EE and DL Medin. 1981. *Categories and Concepts*.

Solomon, KO, DL Medin, and E. Lynch. 1999. “Concepts Do More than Categorize.” *Trends in*

cognitive sciences.

Srinivasan, A. 1999. "The Aleph Manual." *Computing Laboratory, Oxford University* 1.

Tong, Simon. 2001. "Active Learning: Theory and Applications." Stanford University.

Valentine, Stephanie, Francisco Vides, George Lucchese, and David Turner. 2012. "Mechanix: A Sketch-Based Tutoring System for Statics Courses." Pp. 2253–60 in *24th Annual Conference on Innovative Applications of Artificial Intelligence*.

Weinberger, KQ and LK Saul. 2009. "Distance Metric Learning for Large Margin Nearest Neighbor Classification." *Journal of Machine Learning Research* 10:207–44.

Wetzel, Jon. 2014. "Understanding and Critiquing Multi-Modal Engineering Design Explanations."

Wilson, Jason R., Evan Krause, Morgan Rivers, and Matthias Scheutz. 2016. "Analogical Generalization of Actions from Single Exemplars in a Robotic Architecture." *Aamas* (Aamas):1015–23.

Winston, PH. 1970. "Learning Structural Descriptions from Examples." Massachusetts Institute of Technology.

Yin, Panrong, KD Forbus, Jeffrey Usher, Brad Sageman, and BD Jee. 2010. "Sketch Worksheets: A Sketch-Based Educational Software System." Pp. 1871–76 in *Proceedings of the 22nd Annual Conference on Innovative Applications of Artificial Intelligence.*

Yu, Qian, Yongxin Yang, Yi-Zhe Song, Tao Xiang, and Timothy Hospedales. 2015. "Sketch-a-Net That Beats Humans." *BMVC*.

Zhang, H., AC Berg, … M. Maire-Computer Vision and, and Undefined 2006. 2006. "SVM-KNN: Discriminative Nearest Neighbor Classification for Visual Category Recognition." Pp. 2126–36 in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. IEEE.