Using Cortically-Inspired Algorithms for Analogical Learning and Reasoning

Marc Pickett[a], David W. Aha[b]

[a] *NRC/NRL Postdoctoral Fellow, Washington, DC 20375*
[b] *Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory (Code 5510), Washington, DC 20375*

Abstract

We consider the neurologically-inspired hypothesis that higher level cognition is built on the same fundamental building blocks as low-level perception. That is, the same basic algorithm that is able to represent and perform inference on low-level sensor data can also be used to process relational structures. We present a system that represents relational structures as feature bags. Using this representation, our system leverages algorithms inspired by the sensory cortex to automatically create an ontology of relational structures and to efficiently retrieve analogs for new relational structures from long-term memory. We provide a demonstration of our approach that takes as input a set of unsegmented stories, constructs an ontology of analogical schemas (corresponding to plot devices), and uses this ontology to find analogs within new stories in time logarithmic in the total number of stories, yielding significant time-savings over linear analog retrieval with only a small sacrifice in accuracy. We also provide a proof of concept for how our framework allows for cortically-inspired algorithms to perform analogical inference. Finally, we discuss how insights from our system can be used so that a cortically-inspired model can serve as the core mechanism for a full cognitive architecture.

Cortex as Substrate for Cognition & Learning

The neocortex is responsible for much of human intelligence, including sensory perception and higher-level cognition (Rakic, 2009). From studies of neuroscience, Mountcastle (1978) proposed the hypothesis that the human neocortex is essentially the same mechanism repeated many times. That is, Mountcastle hypothesized that higher level cognition is built on the same fundamental building blocks as low-level perception. Such a proposition is attractive from the viewpoint of Artificial Intelligence because it implies that AI researchers need only implement a handful of mechanisms, rather than specialized mechanisms for the myriad aspects of human intelligence (Cassimatis, 2006). For example, under this proposition one wouldn't need to implement separate algorithms for perception and higher-level cognition.

The differences between human brains and the brains of other mammals seems to be quantitative rather than qualitative (Roth & Dicke, 2005). The chief difference between human brains and those of other mammals is that humans have a vastly expanded neocortex (Rilling, 2006). In terms of gross neuroanatomy, human brains seem to have no special structures or mechanisms that are absent in the brains of simpler mammals, such as rabbits, that have little cognitive capacity beyond perception and action. If an expanded neocortex accounts for the bulk of the cognitive differences between humans and other mammals, then an open question is how an expanded neocortex might account for these differences. That is, an account is missing of how a cortical substrate can be leveraged to account for higher level cognition, such as symbolic reasoning and analogical inference (Granger, 2011). This is the larger question that we will partially address in this paper.

In addressing this larger question, we will also address some common criticisms of vector-based approaches to AI (such as connectionism): that they cannot represent (much less learn) relational schemas such as "sibling", and that such systems cannot perform simple parameterized logical inferences such as "If A loves B and B loves C, then A is jealous of C." (Marcus, 1998). We have taken steps to address these criticisms by showing how a second (non-connectionist) system can transform relational data into feature bags (or equivalently sparse fixed-width vectors) such that surface overlap among these feature bags corresponds to structural similarity in the relational data. Unlike related approaches (Socher et al., 2012; Rachkovskij et al., 2013; Levy & Gayler, 2008), our representation can exploit partial analogical schemas. That is, a partial overlap in our representation's feature bags corresponds to a common subgraph in the corresponding structures. With this transform, we recast some processes of higher-level cognition as processes that can be performed by a model inspired by the sensory cortex.

In this paper, we provide background on a simple model loosely based on the sensory cortex, and we show how this model can be leveraged to process relational structures. We claim that this model can be leveraged to perform some functions, such as analogical inference, usually considered to be in the realm of higher cognition. Our approach has the added benefit of yielding an algorithm, called *Spontol*, that addresses the problem of *spontaneous analogy*, a hitherto open problem in computational analogy that asks how analogs can be efficiently parsed, stored, and quickly retrieved from long-term memory (Pickett & Aha, 2013). That is, given a corpus of many large unsegmented relational structures, Spontol discovers analogical schemas that are useful for concisely encoding the corpus and efficiently retrieves analogs given a new structure. For example, given a set of narratives in predicate form, Spontol discovers plot devices and analogs among them.

In the remainder of this paper, we give background on *Ontol* (Pickett, 2011), a model of learning and basic inference inspired by the sensory cortex, describe a novel transform T that converts relational structures into a form on which Ontol can operate, demonstrate the Spontol system, which uses this transform to leverage Ontol to address the problems of analogical retrieval and inference,

discuss implications and shortcomings of our system, and conclude.

Ontol: A Model Inspired by the Sensory Cortex

In this section, we provide background on a model inspired by the human sensory cortices (auditory, visual, tactile) called *Ontol* (Pickett, 2011), that we will use in later sections. Ontol is a pair of algorithms, both of which are given "sensor" inputs (feature bags or fixed-length, real-valued non-negative vectors). The first algorithm, called chunk, constructs an ontology that concisely encodes the inputs. For example, given a set of feature bags representing 50 by 50 windows from natural images, Ontol produces a feature hierarchy similar to that found in the visual cortex. The second algorithm, called parse, takes as input an ontology (produced by the first algorithm) and a new feature bag, and *parses* the feature bag. That is, it produces as output the new feature bag encoded in the higher-level features of the ontology. In addition to "bottom-up" parsing, the second algorithm also makes "top-down" predictions about any unspecified values in the feature bag by recursively flattening the feature hierarchy.

Ontol is ignorant of the modality of its input. It is given no information about what sensory organ or device is producing its inputs. Instead of relying on innate knowledge about a modality, the appropriate features are learned from an ample supply of sensory data. This property allows another system (called *Spontol*) to leverage Ontol to find patterns in abstract "sensory" inputs that are actually encodings of relational structures. Ontol's modality ignorance is biologically inspired. In particular, there is evidence of the plasticity of the sensory cortices of newborn ferrets (and presumably other mammals): When visual data is rerouted into the primary auditory cortex of newborn ferrets, the ferrets' auditory cortex learns features that are similar to those developed in the visual cortex in normal ferrets (Sur & Rubenstein, 2005).

Ontology Learning

Ontol's ontology formation algorithm, called chunk, searches for concepts (or *chunks*) that allow for concise characterization of feature bags. Since chunks themselves are bags of features, chunk is applied recursively to create an ontology. This algorithm is similar to the *recursive block pursuit* algorithm described by Si & Zhu (2011) in that both search for large frequently occurring sets of features. The chunk algorithm differs in that it allows for multiple inheritance, while recursive block pursuit creates only strict tree structures. In the section on *Spontaneous Analogy*, we show the importance of this property for finding multiple analogical schemas within a single relational structure.

The chunk algorithm is shown in Figure 1. It takes as input a set B of feature bags (together with an integer search parameter *samples*) and produces an ontology Ω. For simplicity, we describe the discrete binary version of the algorithm without efficiency modifications, but this can be modified for feature bags with continuous non-negative real values for each feature. The chunk algorithm searches for intersections among existing feature bags and proposes these

```
// Returns an ontology Ω to compress B, a set of feature bags
// samples is the # of candidate concepts at each iteration.
define chunk(B, samples):
   Ω = copy(B)
   while the description length (DL) of Ω is decreasing:
      bestScore = 0
      foreach candidate ∈ getCandidates(Ω, samples):
         score = scoreCandidate(Ω, candidate)
         if score > bestScore: best, bestScore = candidate, score
         if bestScore > 0:
            conceptName = new unique symbol
            Ω = replaceBest(Ω, best, conceptName) ∪ {best}
   return Ω

// Returns samples randomly chosen intersections of bags in Ω
define getCandidates(Ω, samples):
   candidates = {}
   for i = 1 ··· samples:
      a, b = drawRandomPair(Ω)
      candidates = candidates ∪ {a ∩ b}
   return candidates

// The decrease in description length if candidate is used as a concept
define scoreCandidate(Ω, candidate):
   return Σ_{n∈Ω} max (0, conceptScore (n, candidate)) - |candidate|

// Returns the compressed version of Ω using concept best to reduce DL
define replaceBest(Ω, best, conceptName):
   Ω' = copy(Ω)
   foreach Ω'_n ∈ Ω':
      if conceptScore(Ω'_n, best) > 0:
         Ω'_n = replace(Ω'_n, best) + conceptName
   Ω'_conceptName = best
   return Ω'

// The decrease in DL if n were expressed using candidate
define conceptScore(n, candidate):
   // Take into account errors introduced by candidate
   return |n− replace(n, candidate)| - | replace(n, candidate)− n| − 1

// Make n inherit from candidate
define replace(n, candidate):
   return n − candidate
```

Figure 1: **Ontol's Chunking Algorithm for Ontology Learning**

as candidates for new concepts. Each candidate is evaluated by how much it would compress the ontology, then the best candidate is selected and added to the set of feature bags, and the process is repeated until no candidates are found that further reduce the description length of the ontology. Figure 2 shows the ontology constructed by this algorithm when applied to an animal dataset, where base-level features could take on positive, negative, or unstated values (e.g., `fins` and ¬`fins` were both features). Ontol was originally developed as a rough model of the sensory cortex (where visual or audio percepts were modeled

as bit-vectors). In this figure, the "sensory percepts" are the binary features for each animal[1].

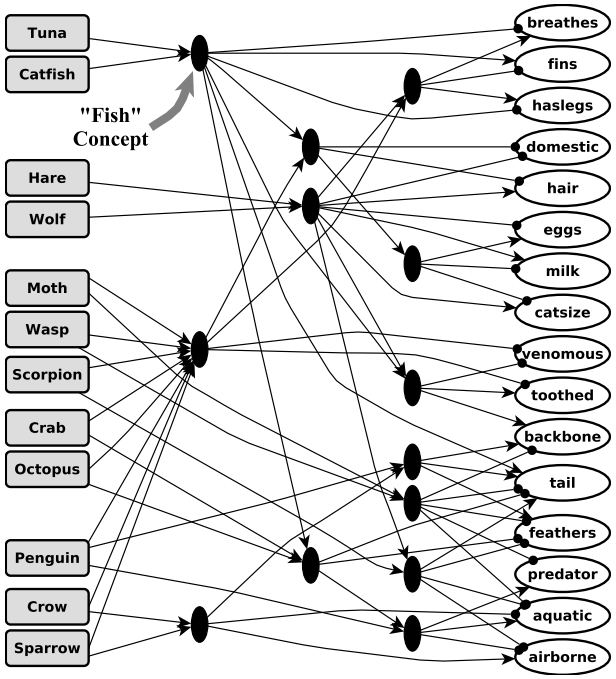


Figure 2: **Part of The Zoo Ontology.** Instances are individual animals shown on the left, and base-level features are on the right. Black nodes in the middle correspond to higher-level features. The concept that corresponds to "fish" is marked. Inhibitory links (for negative features, such as ¬`fins`) are shown as dark circles.

Parsing and Prediction

Given an ontology and a new instance, Ontol's parse(b, Ω) algorithm encodes the feature bag instance b using the higher-level features in the ontology Ω. For example, given the ontology shown in Figure 2 and a new animal (a goldfish) that doesn't breathe, has fins, has no feathers, and is domestic, Ontol will parse the animal as an instance of the *fish* concept[2], with the exception that it is domestic. Ontol's parse algorithm is given in Figure 3. If Ontol is given no other information about the animal, it will also perform top-down inference, and *unfold* the fish concept to predict that the new instance has eggs, no hair, has a tail, etc.. This latter step is called "top-down prediction". Ontol searches for the parse that minimizes the description length of the instance. In our goldfish

[1]Source code for Ontol and other algorithms in this paper can be downloaded at `http://marcpickett.com/src/analogyDemo.tgz`.

[2]Ontol names higher-level concepts with arbitrary tags. We use the term "fish" for simplicity.

example, the raw description of the goldfish consists of 4 features {¬`breathes`, `fins`, ¬`feathers`, `domestic`}, while the compressed encoding has only 2 features {`fish`, `domestic`}. Although the optimal parsing problem is NP-complete (via reduction from 3-SAT), an approximation using a single greedy bottom-up pass can be performed in time logarithmic in the number of learned concepts (Pickett, 2011). Importantly, Ontol examines only a small subset of the concepts and instances while parsing. This means that, when judging concept similarity, Ontol does not need to compare each of its n nodes. This property is important for spontaneous analog retrieval (described below).

```
// Bottom-up greedily parses feature bag b using concepts in Ω.
// parents_j are the concepts in Ω that directly include concept j.
// flat_i is the flattened (non-parsed) representation of concept i.
define parse(b, Ω):
  unexplained = copy(b); b' = {}; bestScore = 1
  // While unexplained is still being explained away.
  while bestScore > 0:
    bestScore = 0
    // Find which concept best explains the remainder.
    foreach i ∈ ⋃_{j∈b'∪unexplained} parents_j:
      if flat_i ⊆ b:
        score = |unexplained ∩ flat_i| − 1
        if score > bestScore: bestScore, best = score, i
    if bestScore > 0:
      // Add best to the parse and remove what best explains.
      b' += best
      unexplained -= flat_best
  // Return the parsed b with what hasn't been explained.
  return b' ∪ unexplained
```

Figure 3: **Ontol's Parsing Algorithm**

Transforming Structures to "Percepts"

We now describe a method for transforming relational structures into feature bags such that the problem of analog retrieval is reduced to the problem of percept parsing. An example of this process is shown for the *Sour Grapes* fable in Figure 4. For this process, we rely on a transform T (described below) that takes a small relational structure and converts it into a feature bag (exemplified in Figure 4(c)). We limit the size of input relational structures for T because T's runtime is quadratic in the size of the structure. We view this limitation as acceptable because people generally cannot keep all the details of an entire lengthy novel (or all the workings of a car engine) in working memory. Generally, people focus on some aspect of the novel, or some abstracted summary of the novel (or engine). Therefore, we randomly break each large relational structure into multiple overlapping *windows*. A window is a small set of connected propositional statements, where two statements are connected if they share at least one argument. By using multiple overlapping windows, we exploit a principle akin to one used by the HMax model of the visual cortex (Riesenhuber &

Poggio, 1999): as the number of windows for a relational structure increases, the probability decreases that another structure has the same windows without being isomorphic to the first.

Related Work on Representing Structure as Features

There has been some work on representing structures as vectors. For example, Holographic Reduced Representations have been used to implement Vector Symbolic Architectures in which there is a correlation between vector overlap and structural similarity (Gayler & Levy, 2009; Rachkovskij et al., 2013). These systems are limited in that they are unable to exploit partial analogical schemas. That is, unlike our representation, a partial overlap in these systems' vectors does not correspond to a common subgraph in the corresponding structures. The ability to represent *partial* structural overlap as partial vector (or feature bag) overlap is important for our system to construct the ontology of analogical schemas that it uses to efficiently retrieve analogs for new structures.

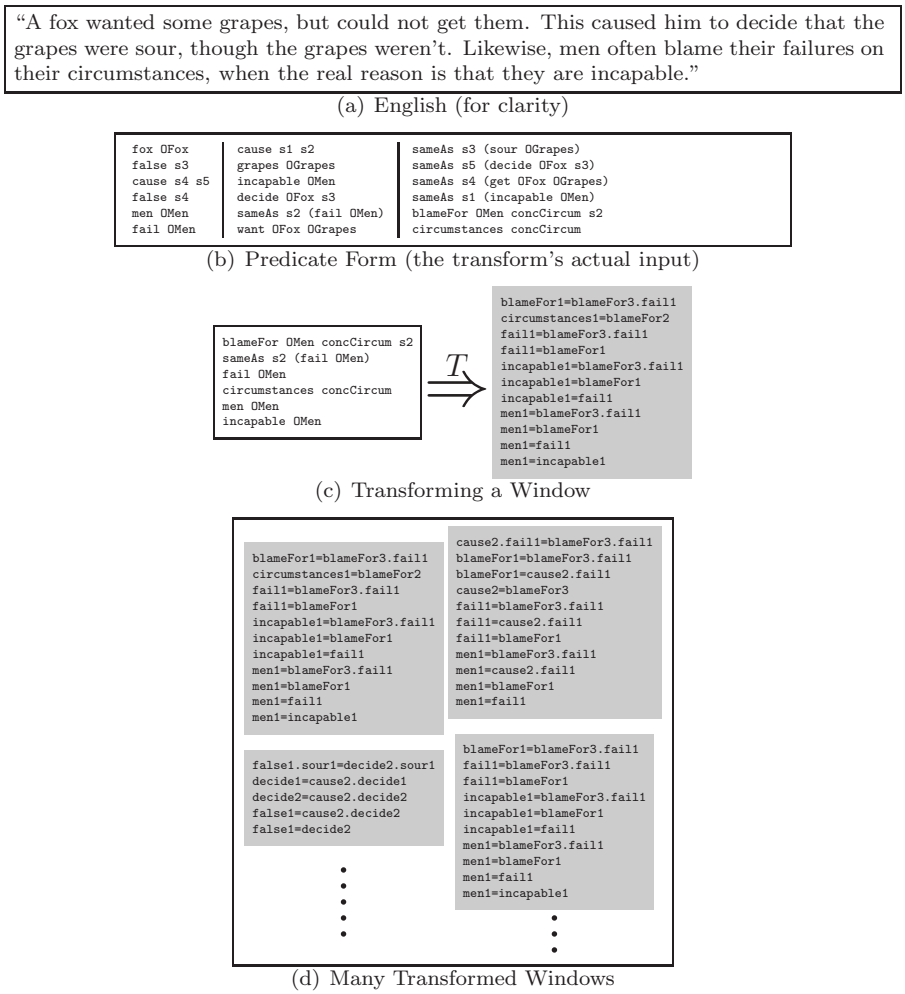
We can also apply `chunk` to feature bag graphlet kernels (Shervashidze et al., 2009), which are related to the transform T below in that both represent partial graphs, but this earlier work applies only for cases where there is one kind of entity, one kind of relation, and only binary relations, while our transform works for multiple kinds of entities and relations, including relations of large arity.

Transforming Small Relational Structures

Here, we describe an operation T, which transforms a (small) relational structure into a feature bag. We consider a relational structure to be a *set* of relational statements, where each statement is either a relation (of fixed arity) with its arguments, or the special relation `sameAs`, which uses the syntax `sameAs <name> (<relation> <arg1> <arg2> ...)`. The `sameAs` relation allows for statements about statements. For example, the statements in Figure 4(b) encode (among other things) that "a fox *decides that* the grapes are sour".

Given a small relational structure s ($\lesssim 10$ statements), T transforms s into a feature bag using a variant of conjunctive coding. That is, T breaks each statement into a set of roles and fillers. For example, the statement `want OFox OGrapes` has two roles and fillers, namely the two arguments of the `want` relation[3]. So T breaks this statement into `want1=OFox` and `want2=OGrapes`, where `want2` means the 2nd argument of `want` (i.e., the "want*ed*"). T then creates one large set of all the roles and their fillers. If there are multiple instances of a relation, T gives them an arbitrary lettering (e.g., `wantB1=OFox`). T makes a special case for the `sameAs` relation. In this case, T uses a *dot* operator to replace the intermediate variable. For example, the statements `sameAs s5 (decide OFox s3)` and `sameAs s3 (sour OGrapes)` would give us `decide2.sour1=OGrapes`. The dot operator allows T to encode nested statements (i.e., statements about statements). Given a set of roles and fillers, T then *chains* the fillers to get

[3]The `O` in `OFox` serves to distinguish this *object* from the unary relation `fox` (where `fox x` means `x` is a `fox`).



(a) English (for clarity)

(b) Predicate Form (the transform's actual input)

(c) Transforming a Window

(d) Many Transformed Windows

Figure 4: **Transforming the *Sour Grapes* Story.** We show the transformation of *Sour Grapes* from predicate form to feature bag form. For clarity, we show an English paraphrase of the story (a), though the input to our transform has already been encoded in the predicate form shown in (b), which shows the story as a set of 18 statements. In (c), we show a window w from the story and its feature bag transform $T(w)$. Finally, the story is represented as many transformed windows (d).

filler equalities. For example, if we have that `want1=OFox` and `decide1=OFox`, then chaining gives us `want1=decide1`. Chaining is essential for recognizing *structural* similarity (as opposed to just surface similarity) between relational structures, and allows us to side-step a criticism of conjunctive coding and tensor products: that the code for `wantB1=OFox` may have no overlap with the code for `want1=OFox` (Hummel et al., 2004). Chaining introduces the code for `wantB1=want1`, which makes the similarity apparent when searching for analogs.

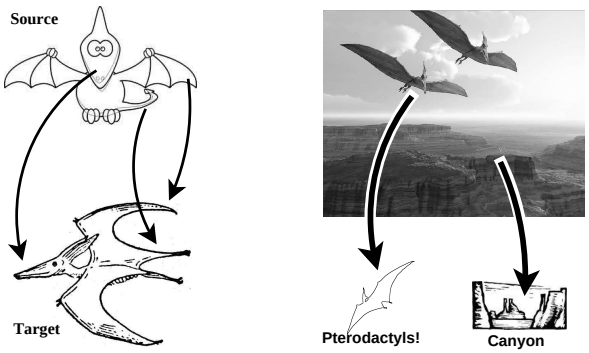
After chaining the roles and fillers, T treats each of these role-filler bindings as an atomic feature. Note that, when we treat roles and fillers as atomic features, Ontol doesn't recognize overlap among feature bags unless they share exactly the same feature. For example, the atomic feature `wantB1=OFox` has no more resemblance to `want1=OFox` for Ontol than it does for any other feature. Also note that the ordering of the roles in each feature is arbitrary but consistent (we use reverse alphabetical order), so there is a `men1=incapable1` feature, but not an `incapable1=men1` feature. The left side of Figure 4(c) shows a window (i.e., a small connected subset) taken from the sour grapes story from Figure 4(b). On the right side is the feature bag transform of this set of 6 statements, consisting of 11 atoms.

Spontaneous Analogy

In our day-to-day experience, we often generate analogies spontaneously (Wharton et al., 1996; Clement, 1987). That is, with no explicit prodding, we conjure up analogs to aspects of our current situation. For example, while reading a story, we may recognize a plot device that is analogous to one used in another story that we read long ago. The shared plot device may be a small part of each story, it is usually not explicitly delineated for us or presented in isolation from the rest of the story, and we may recognize the analogy of the plot device even if the general plots of the two stories are not analogous. Somehow, we *segment* out the plot device and *retrieve* the analog[4] from another story in long-dormant memory. *Spontaneous analogy* is the process of efficiently retrieving an analog from long-term memory given an unsegmented probe structure such that part of the probe shares structural similarity with the analog, though they might not share surface similarity. This process differs from standard models of analogy, which are given a *delineated* probe, and often specify a delineated source analog from which to map. For example, our system is given a large story in its entirety, rather than just a delineated plot device. Given a pair of analogs, analogical mapping is relatively straightforward. The more difficult problem is finding the analogs to begin with. As Chalmers et al. (1992) argue "when the program's discovery of the correspondences between the two situations is a direct result of its being explicitly given the appropriate structures to work with, its victory in finding the analogy becomes somewhat hollow".

The process of spontaneous analogy shares some properties with low-level perception, as exemplified in Figure 5. Within seconds of being presented with a visual image of a pterodactyl flying over a canyon, one can typically describe the image using the word "pterodactyl", even if one has had no special explicit

[4]In our terminology, an *analog* is a substructure of a domain that is structurally similar to a substructure of another domain, and an *analogical schema* is a generalization of an analog. For example, an input domain might be the entire story of *Romeo and Juliet*, an analog would be the part of the story where Romeo kills Tybalt, who killed Romeo's friend, Mercutio (like in *Hamlet* where Hamlet kills Claudius, who killed Hamlet's father), and an analogical schema would be the generalized plot device of a "revenge killing".



(a) Mapping (b) Spontaneous Retrieval

Figure 5: **An analog of Analogical Mapping vs. Spontaneous Analogy.** In Analogical Mapping (a), we are given an explicit source and target, free from interfering context. In spontaneous analogy (b), the analogs (represented by the "Pterodactyl" and "Canyon" concepts) are spontaneously retrieved from long-term memory given an unsegmented probe (represented by the top image).

recent priming for this concept, indeed even if one has not consciously thought about pterodactyls for several years. For us to produce the word "pterodactyl", we must *segment* the pterodactyl from the canyon and retrieve the "pterodactyl" concept from the thousands of concepts stored in memory. We must have learned the "pterodactyl" concept to begin with from unsegmented images. Furthermore, we assume that the brain's mechanism for retrieving the pterodactyl concept is more efficient than exhaustively visiting every concept in long-term memory. It seems unlikely that the representation for "the back door of my kindergarten classroom" is activated while viewing the pterodactyl image, though one might be able to quickly identify an image of this door (or some other specific rarely-visited concept). This perceptual process is robust to noise: The pterodactyl in the image could be partially occluded, ill-lit, oddly colored, or even drawn as a cartoon, and we are still able to correctly identify this shape (to a certain point). Likewise, many details of the plot devices from the above story example could be altered or obfuscated, but this analogy would degrade gracefully.

Related Work on Spontaneous Analogy

There has been earlier work on the problem of analogy in the absence of explicitly segmented domains. The COWARD system of Baldwin & Goldstone (2007) addresses this problem by searching for mappings within a large graph, essentially searching for isomorphic subgraphs. SUBDUE (Holder et al., 1994) compresses large graphs by breaking them into repeated subgraphs, but is limited in that the output must be a strict hierarchy, and would be unable to discover the lattice structure of the concepts in Figure 2. Nauty (McKay, 1981) uses a number of heuristics to efficiently determine whether one graph is a subgraph of another, but it must be given source and target graphs to begin with.

The MAC phase of MAC/FAC (Forbus et al., 1995) bears some relation to our spontaneous analog retrieval. MAC uses vectors of content, such as the number of nodes and edges in a graph, as a heuristic for analog retrieval. However, in cases where the subgraph in question is a part of a much larger graph, the heuristics that MAC uses are drowned out by the larger graph. Furthermore, our system uses "chained" features, which is a core difference between MAC's content vectors and our feature bags. ARCS (Thagard et al., 1990) also assumes that analogs have been delineated (i.e., it matches an entire probe, rather than a substructure). SEQL (Kuehne et al., 2000) generalizes relational concepts, but doesn't build a hierarchical ontology of analogical schemas. Yaner & Goel (2006) describe a two-stage analog retrieval system similar to MAC/FAC, but it differs from our work in that the first (filtering) stage still considers every possible analog (requiring $O(n)$ time in the number of analogs in memory). Below, we show how Spontol builds an ontology that it then uses as an "indexing structure" to retrieve analogs in logarithmic time.

The Conceptual Analogy system of Börner (2001) uses hierarchical clustering on a set of relational structures to learn a hierarchy that it then uses to efficiently find analogs for new structures, which is similar in spirit to our system's use of an ontology for indexing analogs. However, the similarity metric used by Börner's system is based on the number of edges shared between graphs (with identically labeled end-nodes), and thus fails to find isomorphic subgraphs in cases where nodes' names are different between structures.

Spontol*: An Algorithm for Spontaneous Analogy*

Here, we describe *Spontol*[5], an algorithm that uses the transform T to leverage Ontol to build an ontology from a set of relational structures, and uses this ontology to efficiently segment and retrieve analogs for new relational structures. Spontol transforms relational structures into feature bags so that their surface similarity corresponds to the structural similarity of the relational structures. After Spontol has made this transformation, the problem of spontaneous analogy is reduced to the problem of feature overlap, and any of several existing vector-based systems (such as connectionist models) can be used to find and exploit patterns in feature vectors.

The process for building an ontology of analogical schemas from large relational structures, called Spontol-Build, is described in Figure 6. This algorithm extracts $numWindows$ windows from each large relational structure, transforms them into feature bags (exemplified in Figure 4(d)), then chunks these feature bags to create an ontology of windows called $windowOntology$. Spontol-Build then re-encodes the windows by parsing them using this ontology, and re-encodes the larger structures (from which the windows came) as a feature bag of the parsed windows. Finally, Spontol-Build runs another pass of chunking on the re-encoded structures to generate the schema ontology.

[5] *Spontol* is short for "**spont**aneous analogy using the **Ontol** ontology learning and inference algorithm".

```
// Creates an ontology of schemas given a set of structures S.
// numWindows is the # of windows to grab per structure.
// windowSize is the # of statements per window.
define Spontol-Build(S, numWindows, windowSize)
   // Randomly grab windows from each structure,
   // and transform them into feature bag form.
   allWindows = {}
   foreach s ∈ S ; for i = 1, ···, numWindows
      let w_{s,i} = grabConnectedStatements(s, windowSize)
      add T(w_{s,i}) to allWindows
   // Run chunk to generate the window ontology
   windowOntology = chunk(allWindows)
   // Re-encode each structure using the reduced-size windows.
   foreach s ∈ S
      bigWindows_s = {}
      for i = 1, ···, numWindows
         add parse(T(w_{s,i}), windowOntology) to bigWindows_s
   // Run chunk to generate the schema ontology.
   schemaOntology = chunk(bigWindows)
   return schemaOntology, windowOntology
```

Figure 6: **Spontol's Ontology Learning Algorithm**

The process of spontaneous analog retrieval, called Spontol-Retrieve, is given in Figure 7. When given a new relational structure s, we encode s by extracting windows from it, parsing these using the *windowOntology*, then parsing the feature bag representation using the *schemaOntology*. This yields a set of schemas that are contained in s.

```
// Finds analogical schemas for relational structure s.
// ⟨args⟩ =
//     s: the input relational structure
//     schemaOntology: the schema ontology
//     windowOntology: the window ontology
//     numWindows: the # of windows to grab per structure
//     windowSize: the # of statements per window
define Spontol-Retrieve(⟨args⟩)
   // Randomly grab windows from s,
   // transform them into feature bag form,
   // and parse them using the window ontology.
   bag_s = empty feature bag
   for i = 1, ···, numWindows
      w_i = grabConnectedStatements(s, windowSize)
      add parse(T(w_i), windowOntology) to bag_s
   // Parse bag_s, the bag representation of s
   relevantSchemas = parse(bag_s, schemaOntology)
   return relevantSchemas
```

Figure 7: **Spontol's Spontaneous Analogy Algorithm**

Spontaneous Analogy using Spontol

We hypothesize that Spontol is more efficient at retrieving analogs than related approaches, such as MAC/FAC. To partially test this hypothesis, we applied Spontol to a database of 126 stories provided by Thagard et al. (1990). These include 100 fables and 26 plays all encoded in a predicate format, where each story is a set of unsorted statements. An example story in predicate form is shown in Figure 4(b). Note that although the predicates and arguments have English names, our algorithm treated all these as gensyms except for the special `sameAs` relation. In this encoding, the smallest story had 5 statements, while the largest had 124 statements, with an average of 39.5 statements.

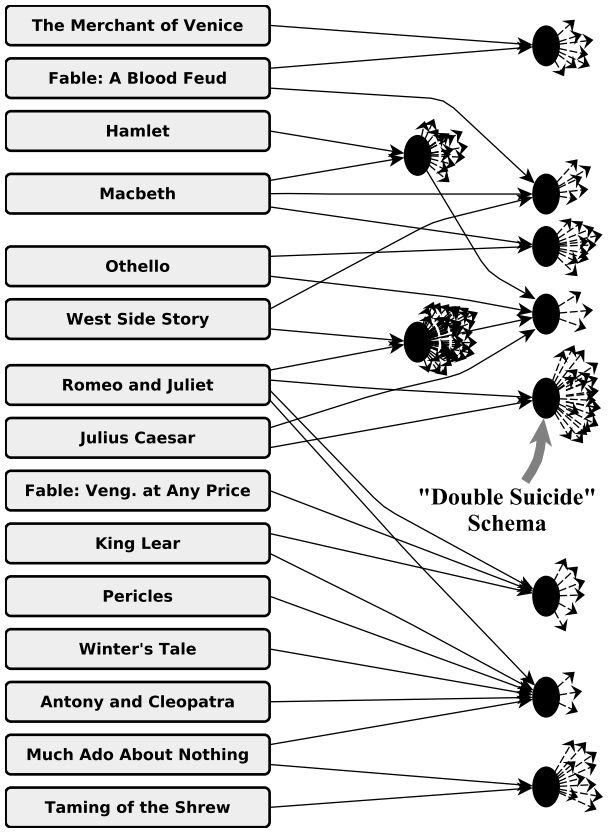


Figure 8: **Part of the ontology Spontol learned from the story dataset.** As in the Zoo Ontology in Figure 2, black ovals represent higher level concepts. The "raw" features (corresponding to the white ovals in Figure 2) are omitted due to space limitations. Instead, we show the outgoing edges from each black oval. While in the Zoo Ontology, the higher level concepts correspond to shared surface features, in this figure, high level concepts correspond to shared structural features, or *analogical schemas*. For example, the denoted oval on the right represents a *Double Suicide* schema, which happens in both *Romeo and Juliet* and in *Julius Caesar*.

We ran Spontol-Build on these stories which produced an ontology of stories,

part of which is shown in Figure 8 (in this case, we somewhat arbitrarily chose $numWindows = 100$ and $windowSize = 20$). This figure shows an analogical schema found in both *Romeo and Juliet* and in *Julius Caesar* that we've labeled as the "Double Suicide" schema. In the first story, Romeo thinks that Juliet is dead, which causes him to kill himself. Juliet, who is actually alive, finds that Romeo has died, which causes her to kill herself. Likewise, in *Julius Caesar*, Cassius kills himself after hearing of Titinius's death. Titinius, who is actually alive, sees Cassius's corpse, and kills himself. The largest schema found (in terms of number of outgoing edges) was that shared by *Romeo and Juliet* and *West Side Story*, which are both stories about lovers from rival groups. The latter doesn't inherit from the Double Suicide schema because Maria (the analog of Juliet), doesn't die in the story, and Tony (Romeo's analog) meets his death by murder, not suicide. Some of the schemas found were quite general. For example, the oval on the lower right with 6 incoming edges and 3 outgoing edges corresponds to the schema of "a single event has two significant effects". And the oval above the Double Suicide oval corresponds to the schema of "killing to avenge another killing".

Spontol-Retrieve uses this schema ontology to efficiently retrieve schemas for a new story, which can be used to make inferences about the new story in a manner analogous to the "goldfish" example from the subsection on *Parsing and Prediction*. To evaluate the efficiency of Spontol-Retrieve, we randomly split our story dataset into 100 training stories and 26 testing stories (we evaluated 100 such partitionings). We then used an ontology learned from the training set, and measured the number of comparisons needed to retrieve schemas (during parse) for the testing set. We compare this approach to MAC/FAC, which, during the MAC phase, visits each of the 100 training stories. Whereas MAC/FAC returns entire stories, Spontol-Retrieve returns analogical *schemas* (just as a visual system might return a generic "pterodactyl" concept rather than specific instances of pterodactyls). For comparison, we modify Spontol-Retrieve to return the set of instances that inherit from $relevantSchemas$, rather than just the schemas.

	Accuracy	**Avg. # Comparisons**
MAC/FAC	100.00% ± .00%	100.00 ± .00
Spontol	95.45% ± .62%	15.43 ± .20

Table 1: **Speed/Accuracy Comparison of Spontol**

Results are shown in Table 1, averaged over 100 trials. We show accuracy (and standard error) for both systems measured as the percentage of stories correctly retrieved, where a story was determined to be correct if it was retrieved by MAC/FAC. Whereas MAC/FAC's case-by-case comparison requires a linear number of operations (in the number of structures), Spontol requires only logarithmic number of comparisons at a slight cost of accuracy. Therefore, Spontol requires an order of magnitude fewer comparisons than MAC/FAC, *or any linear look-up algorithm* (for a survey, see Rachkovskij et al. (2013)). For larger

datasets, we hypothesize that these differences will be even more pronounced. Although each comparison by both MAC and Spontol-Retrieve is a fast vector operation, for very large datasets (e.g., 10^9 relational structures), even a linear number of vector operations becomes impractical.

Our comparison doesn't take into account the computational overhead used by Spontol in building the schema ontology (whereas MAC/FAC doesn't require this overhead). While this overhead is currently significant, we are developing an incremental version of chunk that builds an ontology in $O(n \log n)$ in the number of feature-bags. Furthermore, the overhead to build the schema ontology *per probe* will be minimal when the number of probes is much larger than the number of stories used to build the ontology.

To test the importance of chaining that Spontol's T transformation uses, we performed the same experiment as above with the exception that the chaining step was skipped. As expected, this caused Spontol to retrieve only those analogs that had some surface similarity to the stories in the testing set. Though the average retrieval time for this version ($10.88 \pm .24$ comparisons) was slightly faster than the full version of Spontol, the accuracy was significantly worse, an average of only $49.64\% \pm .80\%$ of the relevant analogs were retrieved.

In future work, we will test these systems on a broader range of relational datasets to help elucidate the conditions under which Spontol yields high accuracy and very-low retrieval cost.

Analogical Inference

Parsing and top-down prediction may be used together with a *chaining* algorithm to perform rudimentary logical inference. Briefly, the chaining algorithm chains *bindings* where a binding is a symmetrical relation stating that two variables have the same value. If A is bound to B, and B is bound to C, then chaining infers that A is bound to C. A simplified example of inference using parsing, top-down prediction, and chaining is shown in Figure 9. In this example, Spontol has learned analogical schemas from stories of theft, diplomatic visits, and defaulted loans. In The Story of Doug, Spontol is told that Doug loaned a spatula to Gary who then defaulted. Spontol parses this story, uses top-down prediction, and chaining to infer that the spatula was lost. This example is simplified in that it does not use windowing, but it shows the basic mechanism of inference.

Discussion

In this paper, we have introduced, demonstrated, and given an initial empirical exploratory analysis for a system that solves the problem of spontaneous analogy. By representing relational structures as feature bags, as described in the section on *Spontaneous Analogy*, we reduce the problems of analogy to problems of surface similarity. Some of these problems and their reduced versions are shown in Figure 10.

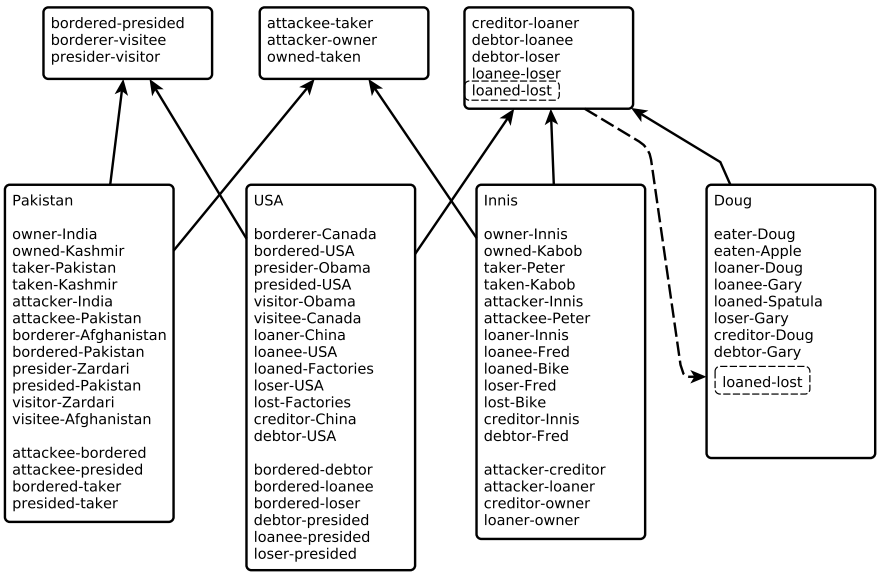


Figure 9: **Basic inference using bottom-up parsing, top-down prediction, and chaining** In this simplified example, we use an ontology of schemas (learned from stories shown on the lower left) to parse The Story of Doug, which is parsed to inherit from the concept at the top-right. This concept has the atomic feature "loaned-lost", which, through top-down implication, we infer to be part of The Story of Doug. We then use our chaining algorithm system to interpret the features in the Story of Doug as bindings, and chain "loaned-lost" with "loaned-Spatula" to infer "lost-Spatula" (i.e., the Spatula was lost).

Problem in Relation Space	**Problem in Feature Bag Space**
Structural Similarity	→ Surface Similarity
Analogical Schema Induction	→ Concept Discovery
Spontaneous Analogical Reminding	→ Concept Recognition
Analogical Inference	→ Top-down Prediction
Structural Segmentation	→ Concept Parsing

Figure 10: **Reducing problems from relational space to feature bag space.**

Our representation also offers a new solution for the *binding problem* for long-term (static) memory that allows for efficient analog retrieval in the absence of explicitly segmented domains. The binding problem asks how we can meaningfully represent bindings between roles and fillers. Most solutions to the binding problem in connectionism (e.g., LISA (Hummel & Holyoak, 2005)) do so in terms of temporal synchronicity, which requires continual activation and is therefore impractical for static memory. Temporal synchronicity only works for knowledge in *working* memory, and these models typically address storage in long-term memory by relying on some form of conjunctive coding or tensor products. Though these systems fail to address how relational structures can

be efficiently retrieved from long-term memory, we hypothesize that a working-memory system, such as LISA, may be necessary for the "chaining" process on which our system relies (though there has also been work on using Vector Symbolic Architectures to chain variables (Kanerva, 2010)).

On a more conceptual level, Spontol makes headway into the deeper problem of the unification of perceptual and cognitive processes. A criticism of "symbolic" approaches to artificial intelligence is the separation of perception and cognition (Chalmers et al., 1992). For example, many cognitive architectures (e.g., SOAR, ACT-R, and ICARUS (Langley et al., 2009)) assume that a perceptual system provides symbols with which to do "cognitive" processes such as planning or analogical reasoning. On the other hand, until now, there have been few accounts of how a perceptual or vector-based system (such as a connectionist network) can efficiently retrieve analogs from long-term memory. Our approach demonstrates how a vector-based system can be used to perform processes formerly only performed by symbolic reasoners.

Future Extensions

Although Spontol addresses some outstanding problems in Computational Analogy, there is still ample room for future work. The most exciting extension to our work is the implementation of other cognitive processes, such as a hypothetical reasoning system, that leverages a cortically-inspired model as its core mechanism for representation, learning, and basic inference. This may potentially provide the flexibility and robustness of a connectionist system while maintaining the combinatorial power of symbolic approaches to Artificial Intelligence.

Although we specify a particular perceptual model, since a feature bag can be represented by a sparse fixed-length vector, our system can easily be extended to instead use any modality-ignorant model that is able to create and use an ontology from sparse fixed-length vectors (Si & Zhu, 2011; Le et al., 2012; Riesenhuber & Poggio, 1999; George & Hawkins, 2009). In future work, we plan to investigate using other vector-based systems to process the feature bags produced by the transform T. In particular, we are currently extending Ontol's chunk and parse algorithms into a single algorithm that is fed input vectors incrementally and assimilates (or parses) them and accommodates (or modifies the ontology) for what it can't assimilate, in a style similar to the theory described by Piaget (1954).

While chunk only finds conjunctions, some of the perceptual models listed above do "pooling" (finding useful *dis*junctions) as well as finding conjunctions (though not in a domain-ignorant way). Pooling has been shown as a means for efficiently representing invariant concepts in perception (e.g., visual objects with invariance to translation, rotation, and scale (Riesenhuber & Poggio, 1999)). If a modality-ignorant perceptual system is developed that finds useful disjunctions, it would be interesting to apply this system to transformed relational structures. We hypothesize that this new version of Spontol would be able to discover relational equivalence classes. For example, our system currently sees no similarity between the symbols `likes` and `loves`, though these symbols are

interchangeable in some cases. We hypothesize that pooling would allow Spontol to exploit this equivalence.

Our implementation for representing a relational structure as a set of windows might not scale well to very large structures without some modifications. An open problem is how windows might be managed in a sensible way. Spontol currently uses "bags of windows" for medium-sized structures. We propose extending Spontol by allowing hierarchies of progressively higher-order bags to represent larger structures (e.g., bags of bags of bags of windows).

A common criticism of conjunctive coding and tensor products is that they cause an explosion of features (Hummel et al., 2004). For example, in our story demonstration, the feature-bag representation used 48,848 atomic features (such as `men1=incapable1`) to represent the bindings from 126 stories, which originally used a total of 3,572 atomic symbols to express their relational forms. To address this, we "aliased" the features that represent bindings by creating an arbitrary many-to-one hash to reduce the number of features from 48,848 to 5,000 (e.g., `men1=incapable1`, `king1=banish2`, `dislike2=causeB2.reject2` and 7 other features all get mapped to `hash1977`). This resulted in some loss of performance: when running the same experiment as before with this change, the average retrieval time increased from 15.43 to 21.73 ± 0.31 comparisons, and the average accuracy decreased from 95.45% to $86.06\% \pm 0.36\%$. We suspect that aliasing causes stories to appear more similar to each other, which causes the number of comparisons to increase. In future work, we plan to further investigate the effect of feature count on speed and accuracy.

In a more cognitively plausible representation, each of the 3,572 atomic symbols would be encoded by a bag of features. That is, our implementation of Spontol currently treats roles and fillers as atoms. Because of this, Spontol fails to find structural similarity when roles are similar, but not identical. For example, Spontol would see no overlap between a "revenge killing" and a "revenge beating" because it sees no similarity between a "killer" and a "beater". A future extension to Spontol is to allow both roles and fillers themselves to be feature bags, which would allow surface similarity between "killer" and "beater". Bindings would then be tensor products of these feature bags.

An important open problem is how relational structures arise from sensor data that isn't explicitly relational. That is, how do people extract entities and relations from raw percepts, which are essentially feature bags representing sensor readings? The stories in our demonstration were already summarized and encoded in predicate logic by a person. A person can watch a video of a production of Romeo and Juliet —a stream of pixels and audio— and produce this summary. How people do this (or how other intelligent systems might do this) is the subject of our longer-term future work.

Conclusion

The chief contribution of this paper is a system, Spontol, that uses the same algorithm to process sensory and higher-level (relational) data. We demonstrated Spontol by using it to solve the problem of spontaneous analogy. That

is, we have demonstrated how Spontol can efficiently store and retrieve analogs without the need for human delineation of schemas.

Spontol may offer evidence in support of the computational feasibility of a uniform "substrate" of intelligence (Mountcastle, 1978). In particular, we've shown how a system that was designed to process perceptual data (Ontol) can be leveraged to process "symbolic" data (i.e., relational structures). This may provide insight into how a model of the sensory cortex may be used as the core mechanism for a full cognitive architecture.

Acknowledgements

We would like to thank all the reviewers for their helpful suggestions. This research was sponsored by NRL. Marc Pickett performed this work while supported by an NRC postdoctoral fellowship at the Naval Research Laboratory. The views and opinions contained in this paper are those of the authors, and should not be interpreted as representing the official views or policies, either expressed or implied, of NRL or the DoD.

References

Baldwin, D., & Goldstone, R. L. (2007). Finding analogies within systems: Constraints on unsegmented matching. In *Workshop on Analogies: Integrating Multiple Cognitive Abilities*.

Börner, K. (2001). Efficient case-based structure generation for design support. *Artificial Intelligence Review*, *16*, 87–118.

Cassimatis, N. L. (2006). A cognitive substrate for achieving human-level intelligence. *AI Magazine*, *27*, 45–56.

Chalmers, D. J., French, R. M., & Hofstadter, D. R. (1992). High-level perception, representation, and analogy: A critique of artificial intelligence methodology. *Journal of Experimental & Theoretical Artificial Intelligence*, *4*, 185–211.

Clement, J. (1987). Generation of spontaneous analogies by students solving science problems. In *Thinking Across Cultures* (pp. 303–308).

Forbus, K., Gentner, D., & Law, K. (1995). MAC/FAC: A model of similarity-based retrieval. *Cog. Sci.*, *19*.

Gayler, R., & Levy, S. (2009). A distributed basis for analogical mapping. In *New Frontiers in Analogy Research; Proceedings of the 2nd International Analogy Conference*.

George, D., & Hawkins, J. (2009). Towards a mathematical theory of cortical micro-circuits. *Computional Biology*, *5*.

Granger, R. (2011). How brains are built: Principles of computational neuroscience. *Cerebrum; The Dana Foundation*, *Available at http://dana.org/news/cerebrum/detail.aspx?id=30356*.

Holder, L., Cook, D., & Djoko, S. (1994). Substructure discovery in the subdue system. In *Workshop on Knowledge Discovery in Databases*.

Hummel, J. E., & Holyoak, K. J. (2005). Relational reasoning in a neurally plausible cognitive architecture: An overview of the lisa project. *Current Directions in Psychological Science*, *14*, 153–157.

Hummel, J. E., Holyoak, K. J., Green, C., Doumas, L. A. A., Devnich, D., Kittur, A., & Kalar, D. J. (2004). A solution to the binding problem for compositional connectionism. In *AAAI Fall Symposium on Computational Connectionism in Cognitive Science*.

Kanerva, P. (2010). What we mean when we say "what's the dollar of mexico?": Prototypes and mapping in concept space. In *2010 AAAI Fall Symposium Series*.

Kuehne, S., Forbus, K., Gentner, D., & Quinn, B. (2000). SEQL: Category learning as progressive abstraction using structure mapping. In *Proceedings of the 22nd Annual Meeting of the Cognitive Science Society*.

Langley, P., Laird, J. E., & Rogers, S. (2009). Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, *10*, 141–160.

Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J., & Ng, A. (2012). Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*.

Levy, S. D., & Gayler, R. (2008). Vector symbolic architectures: A new building material for artificial general intelligence. In *Proceedings of The First Conference on Artificial General Intelligence*.

Marcus, G. F. (1998). Rethinking eliminative connectionism. *Cognitive Psychology*, *37*, 243–282.

McKay, B. (1981). Practical graph isomorphism. *Congressus Numerantium*, *30*, 45–87.

Mountcastle, V. (1978). An organizing principle for cerebral function: The unit model and the distributed system. *The Mindful Brain*, (pp. 7–50).

Piaget, J. (1954). *The Construction of Reality in the Child*. Basic Books.

Pickett, M. (2011). *Towards Relational Concept Formation From Undifferentiated Sensor Data*. Doctoral dissertation University of Maryland Baltimore County.

Pickett, M., & Aha, D. (2013). Spontaneous analogy by piggybacking on a perceptual system. In *Proceedings of the 35th Annual Conference of the Cognitive Science Society*.

Rachkovskij, D., Kussul, E., & Baidyk, T. (2013). Building a world model with structure-sensitive sparse binary distributed representations. *Biologically Inspired Cognitive Architectures*, *3*, 64–86.

Rakic, P. (2009). Evolution of the neocortex: A perspective from developmental biology. *Nature Reviews Neuroscience*, *10*, 724–735.

Riesenhuber, M., & Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nature Neuroscience*, *2*, 1019–1025.

Rilling, J. K. (2006). Human and nonhuman primate brains: Are they allometrically scaled versions of the same design? *Evolutionary Anthropology: Issues, News, and Reviews*, *15*, 65–77.

Roth, G., & Dicke, U. (2005). Evolution of the brain and intelligence. *Trends in Cognitive Sciences*, *9*, 250–257.

Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., & Borgwardt, K. (2009). Efficient graphlet kernels for large graph comparison. In *International Conference on AI & Statistics*.

Si, Z., & Zhu, S. (2011). Unsupervised learning of stochastic and-or templates for object modeling. In *IEEE International Conference on Computer Vision Workshops* (pp. 648–655). IEEE.

Socher, R., Huval, B., Manning, C. D., & Ng, A. Y. (2012). Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Sur, M., & Rubenstein, J. L. (2005). Patterning and plasticity of the cerebral cortex. *Science Signaling*, *310*, 805.

Thagard, P., Holyoak, K., Nelson, G., & Gochfeld, D. (1990). Analog retrieval by constraint satisfaction. *Artificial Intelligence*, *46*, 259–310.

Wharton, C., Holyoak, K., & Lange, T. (1996). Remote analogical reminding. *Memory & Cognition*, *24*, 629–643.

Yaner, P. W., & Goel, A. K. (2006). Visual analogy: Viewing analogical retrieval and mapping as constraint satisfaction problems. *Applied Intelligence*, *25*, 91–105.