Advances in Cognitive Systems X (20XX) 1-16 Submitted X/20XX; published X/20XX

TRESTLE: Incremental Learning in Structured Domains using Partial Matching and Categorization

Christopher J. MacLellan CMACLELL@CS.CMU.EDU
Erik Harpstead EHARPSTE@CS.CMU.EDU
Vincent Aleven ALEVEN@CS.CMU.EDU
Kenneth R. Koedinger KOEDINGER@CMU.EDU
Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA

Abstract

We present TRESTLE, an incremental algorithm for probabilistic concept formation in structured domains that builds on prior concept learning research. TRESTLE works by creating a hierarchical categorization tree that can be used to predict missing attribute values and cluster sets of examples into conceptually meaningful groups. It is able to update its knowledge by partially matching novel structures and sorting them into its categorization tree. The algorithm supports mixed-data representations, including nominal, numeric, relational, and component attributes. We evaluate the algorithm's performance on prediction and categorization tasks and show preliminary evidence that this new categorization model is competitive with non-incremental approaches and more closely approximates human performance.

1. Introduction

Humans have the ability to improve their performance with experience. In order to better understand these capabilities, there have been numerous research efforts to construct computational models of human learning (Vanlehn, Jones, & Chi, 1992; Fisher & Yoo, 1993; Li et al., 2012; Laird, Rosenbloom, & Newell, 1986; Langley, Laird, & Rogers, 2008). Early work on human learning embraced categorization as a primary mechanism for organizing experiences, recalling them in new situations, and using them to make decisions (Feigenbaum & Simon, 1984; Fisher & Yoo, 1993). In contrast, most recent work in cognitive architectures emphasizes the generation of solutions to problems or the execution of actions (e.g., planning and speed-up learning) (Langley, Laird, & Rogers, 2008). However, categorization and conceptual understanding remain crucial aspects of cognition. For example, there is evidence that humans spend more time on learning to recognize the conditions for an action than on learning the steps needed to perform it (Zhu et al., 1996). Thus, we argue that more research should be conducted to better understand human categorization and the role it plays in learning and problem solving.

One important aspect of human categorization is that it occurs in an incremental fashion, enabling humans to improve their performance given more experience and to adapt their knowledge and understanding in response to novel situations. This format of learning is in contrast to the

main thrust of research in the field of machine learning, which tends to explore non-incremental approaches (e.g., support vector machines and decision trees) in an effort to achieve optimal performance. Thus, while incremental learners are able to more quickly update their knowledge given new experiences, their performance is affected by the order in which these experiences occur. For example, when humans are learning to solve algebra equations the order in which they receive practice problems affects their learning. In fact, how best to order problems for human learners remains an open research question that would stand to benefit from computational models of incremental learning (Li, Cohen, & Koedinger, 2012; Carvalho & Goldstone, 2013).

A second characteristic of human categorization is that it occurs in a wide range of environments and is affected by different types of information; i.e., structural, relational, nominal, and numeric information. For example, the work of Carvalho and Goldstone (Carvalho & Goldstone, 2013) has shown that the structure of examples affects human concept formation. Other work exploring how humans form analogies has identified structure mapping as a crucial ability that allows humans to map their prior concepts to new situations (Forbus, Gentner, & Law, 1995; Holyoak, 2005). While prior models of concept learning have explored how to handle relational (Quinlan & Cameron-Jones, 1995), nominal (Fisher, 1987), and numeric (McKusick & Thompson, 1990; Li & Biswas, 2002; Quinlan, 1986) information, less emphasis has been placed on learning with structural information (Thompson & Langley, 1991; Forbus, Gentner, & Law, 1995). Further, few studies exist exploring how best to integrate multiple data types; of these, only a subset integrate nominal and numeric data (Li & Biswas, 2002; Quinlan, 1986; McKusick & Thompson, 1990). Even fewer studies integrate relational (Quinlan & Cameron-Jones, 1995) or structural (Thompson & Langley, 1991; Forbus, Gentner, & Law, 1995) information. We claim that any system attempting to model human categorization should account for and integrates all of these data types into a single learning model.

A third characteristic of human concept learning is that it occurs in both supervised and unsupervised settings. In many cases, these two modes of learning are treated as distinct; however, there is evidence that humans can improve their performance on unsupervised tasks given supervision on a different task (Zhu et al., 2007), suggesting that humans use shared knowledge for these two tasks. Further, humans have the capacity to use learned concepts in a generative fashion to perform pattern completion or to flexibly predict missing attributes of new examples, even if they are not told these attributes during training time. For example, Gestalt psychologists discovered that humans have a tendency to see completed form when presented with incomplete geometric shapes, a term they refer to as the principle of closure (Todorovic, 2008). This finding suggests that models of human categorization should be capable of operating with and without supervision and that the knowledge used for these tasks influence each other.

To explore these aspects of human categorization, we developed TRESTLE, an incremental algorithm for probabilistic concept formation in structured domains that builds on prior research in categorization modeling (Fisher, 1987; McKusick & Thompson, 1990; Thompson & Langley, 1991). This approach is able to map novel structures to its existing knowledge in an online fashion and update its categorization knowledge based on these new structures. TRESTLE can also handle mixed representations, allowing it to function with nominal, numeric, relational, and component attributes. Finally, this approach supports partial matching, so it can process incomplete or partially specified input. These features allow TRESTLE to use its categorization knowledge for predicting

Figure 1: A screenshot of a player building a tower in *RumbleBlocks*. The final tower must cover the light blue energy balls and survive a simulated earthquake to be successful.

missing attributes in a supervised way and for clustering examples in purely unsupervised situations. In this paper, we present *RumbleBlocks*, an example domain containing nominal, numeric, relational, and structured information; describe TRESTLE's approach for learning concepts in this domain; and present a preliminary evaluation of the algorithm comparing it with more specialized approaches as well as with humans. We show that TRESTLE is able to learn probabilistic concepts and make predictions based on these concepts at a human level of performance. Additionally, we inspect TRESTLE's conceptual organization and show that it has reasonably high agreement with human clusterings. These results provide an initial demonstration of how TRESTLE can be used to model human concept formation in structured domains.

2. Example Domain: *RumbleBlocks*

One domain we have explored for the purpose of modeling human learning is *RumbleBlocks* (Christel et al., 2012), an educational game designed to teach concepts of structural stability and balance to young children. What makes this domain interesting is the detailed structural quality of game tasks. In the game, players are tasked with building a tower out of blocks in a two-dimensional, continuous, world with a realistic physics simulation. Players must design their towers to cover a series of energy balls in an attempt to power an alien's spaceship (see Figure 1). These energy balls function as both scaffolding and constraints on players' designs. Once they are done designing their tower, players place the spaceship on the tower, which charges the ship and causes an earthquake. If the tower survives the earthquake with the spaceship intact, then the player succeeds and goes on to the next level, otherwise they fail and have to try the level again.

Each level of the game is designed to emphasize one of three primary concepts of structural stability and balance: objects with wide bases are more stable, objects with lower centers of mass are more stable, and symmetrical objects are more stable. While the steps of how to construct a tower are necessary to succeed in the game (e.g., the actions necessary to drag a block across the

screen), they are not relevant to the educational goals of its design. The central learning challenge for players of *RumbleBlocks* is to understand how to categorize the states of the game world as good or bad examples of the game's target concepts.

RumbleBlocks provides a number of interesting challenges when attempting to model players' conceptual learning. The biggest challenge is in representing the richly structured space of the game. Game states exist in a continuous space and possess many important structural features (e.g. different types of blocks might be used to construct an arch or other high level pattern). In addition to representational issues, the physics simulation that models the earthquake in the game is susceptible to minor perturbations in rigid body physics. This means that two player towers that appear outwardly similar have a small chance of being treated differently in terms of the success criteria, resulting in noisy evaluation feedback.

An early attempt exploring how players learn in *RumbleBlocks* sought to model the acquisition of structural patterns as a grammar induction task through a process called Conceptual Feature Extractions (CFE) (Harpstead et al., 2013). CFE takes a series of examples and discretizes them to a grid before exhaustively generating a two-dimensional context-free grammar that is capable of parsing each example in every possible way (i.e., a combinatorial number of parse trees for each example). The parse trees for each example are then converted into feature vectors using a bag of words approach, where each symbol in the grammar has its own feature. The generated feature vectors can then be used in traditional machine learning approaches for supervised prediction or unsupervised clustering. The goal of this approach was to construct features that capture the maximal amount of structural information of the examples so that this information could be used in learning.

While the CFE approach was successful for clustering student solutions in order to understand how players' design patterns differed from those expected by the game's designers (Harpstead et al., 2013), it has not been evaluated as part of a model of human categorization in *RumbleBlocks*. Further, a number of features of this approach present issues in terms of its ability to perform this task. Firstly, the algorithm runs in a batch fashion instead of incrementally and requires all examples from the entire game to generate its grammar. Thus, any new player examples that posses some new unseen feature would require the learning of an entirely new grammar. Secondly, the grammar rule formalism used by CFE begins to break down in continuous environments and when two-dimensional objects cannot be easily decomposed across axis (e.g., a spiral shape of blocks). Finally, CFE, and other similar grammar approaches (Talton et al., 2012), cannot handle missing data; i.e., they cannot partially match during parsing. So, if an attribute is missing from the the training set, then examples containing that attribute cannot be parsed during runtime. Similarly, if an attribute is always present in training but not at runtime, then there are situations where the examples would not be parsable.

3. TRESTLE

In order to better model human concept formation in the *RumbleBlocks* domain we developed TRESTLE.[1] This approach incrementally constructs a categorization tree given the sequential pre-

1. TRESTLE source code is available at: `http://github.com/cmaclell/concept_formation`

sentation of instances. This categorization tree can then be used to make predictions about the given instances or to generate clusterings. At a high level, the TRESTLE algorithm proceeds through three major steps when given an instance:

1. **Partial matching**, which structurally renames the instance to align it with existing concepts.
2. **Flattening**, which converts a structured instance into an unstructured one while preserving its structural information with a specific naming scheme.
3. **Categorization**, which incorporates the example into existing knowledge.

In the following sections we describe how TRESTLE represents both instances and concepts. We then describe the processing steps that TRESTLE performs to learn from each instance.

3.1 Instance and Concept Representations

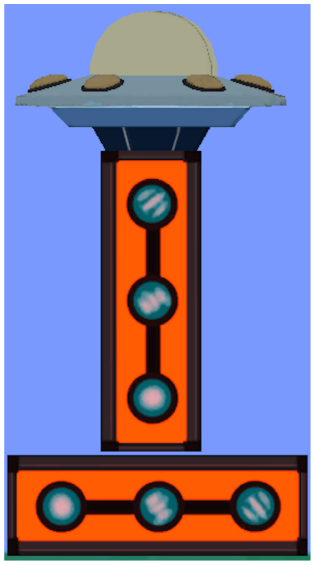
The TRESTLE algorithm uses two representations to perform learning: one for instances (i.e., specific examples it encounters) and one for concepts (i.e., collections of examples in memory). Instances are represented in TRESTLE as sets of attribute values, where each attribute can have one of four types: **nominals**, which represent discrete information (e.g., the type or color of a block); **numerics**, which represent continuous information (e.g., the position of a block in continuous space); **components**, which represent sub-elements of instances (e.g., a block with its own set of attribute values is a component of a greater tower); and **relations**, which can be used to represent relationships between components (e.g., two vertically adjacent blocks). Figure 2 shows an example of a *RumbleBlocks* tower and how it is represented using these four attribute-value types. In this example, we include the "On" relation to demonstrate how relations are represented, but in practice the component information is often sufficient for learning meaningful concepts. In fact, our subsequent evaluation of TRESTLE in the *RumbleBlocks* domain did not require the hand generation of relations, we only utilized the component and success information that was automatically provided by the game engine. Also, the current example only shows component attributes in the greater instance, but sub-instances stored in component attributes can also contain further component attributes.

In response to being presented with instances, TRESTLE forms a hierarchy of concepts, where each concept is a probabilistic description of the collection of instances stored under it. This probabilistic description is stored in the form of a probability table that can be used to track the counts of how often each attribute value occurs in the underly instances (e.g., see Figure 2). These probability tables can be used to compute the probability of different attribute values occurring in an instance given its concept label. Additionally, the concept maintains a count of the number of instances it contains, so that the probability of the concept given its parent concept can be computed.

3.2 Partial Matching

When presented with an instance, TRESTLE starts by partially matching the instance to the root concept. This process consists of renaming any component attributes in the instance to increase the similarity of the instance to the root concept. For example, Component1 in the instance shown in Figure 2 might be renamed to C1 if that is the name most commonly assigned to a UFO block with similar coordinates in the root concept. The other components would also be renamed to further

Game State



Instance Description

```
{ Successful: "False",
  Component1: { type: "ufo",
                angle: 0.0,
                left: 0.1,
                right: 2.8,
                bottom: 4.1,
                top: 5.6 },
  Component2: { type: "rectangle",
                angle: 90.0,
                left: 0.9,
                right: 1.9,
                bottom: 1.1,
                top: 4.1 },
  Component3: { type: "rectangle",
                angle: 0.0,
                left:  0.0,
                right: 3.0,
                bottom: 0.0,
                top: 1.0 },
  (On Component1 Component2): "True",
  (On Component2 Component3): "True}
```

Concept Description

```
{ # instances = 10,
  P(Successful) = {"False": 0.7, "True": 0.3},
  P(C1.type) = {"ufo": 1.0},
  P(C1.angle) ~ N(0.0, 0.1),
  P(C1.left) ~ N(0.1, 0.2),
  P(C1.right) ~ N(2.8, 0.2),
  P(C1.bottom) ~ N(4.1, 0.1),
  P(C1.top) ~ N(5.6, 0.1),
  P(C2.type) = {"rectangle": 1.0},
  P(C2.angle) ~ N(90.0,0.1),
  P(C2.left) ~ N(0.9, 0.3),
  P(C2.right) ~ N(1.9, 0.3),
  P(C2.bottom) ~ N(1.1, 0.1),
  P(C2.top) ~ N(4.1, 0.1),
  P(C3.type) = {"rectangle": 1.0},
  P(C3.angle) ~ N(0.0, 0.1),
  P(C3.left) ~ N(0.0, 0.2),
  P(C3.right) ~ N(3.0, 0.2),
  P(C3.bottom) ~ N(0.0, 0.1),
  P(C3.top) ~ N(1.0, 0.1),
  P((On C1 C2)) = {"True": 1.0},
  P((On C2 C3)) = {"True": 1.0}}
```

Figure 2: A tower in *RumbleBlocks*, its representation as an instance in TRESTLE using the four attribute-value types (nominal, numeric, component, and relational), and the representation of a TRESTLE concept that might describe the instance. The concept stores the number of instances categorized as this concept, the probability of each nominal attribute value given their occurrence counts, and the normal density function for each numeric attribute given the mean and standard deviation of their values.

increase similarity. When component attributes are renamed, any relation attributes that reference them are also updated. In order to evaluate similarity between the instance and the concept, TRESTLE computes the expected number of attribute values that it can correctly guess after incorporating the matched instance into the root concept and chooses a match that maximises this value. Under the assumption that an attribute A_i with value V_{ij} can be guessed with the probability $P(A_i = V_{ij})$ and is correct with the same probability, maximizing the similarity is equivalent to maximizing: $\sum_i \sum_j P(A_i = V_{ij})^2$. This optimization function can be efficiently computed using only the root concept's probability table.

To select the best match, TRESTLE searches the space of all matches using the A* algorithm. To heuristically evaluate the quality of each state in this search TRESTLE computes the change in the number of expected correct guesses for the component attributes it has already matched as well as the best possible improvement in expected correct guesses for unmatched component attributes. This heuristic is admissible and ensures that the best possible match is found.

3.3 Flattening

After matching the instance to the root concept, TRESTLE then flattens the instance using two procedures. First, it converts all relational attributes directly into nominal attributes. Second, com-

ponent attributes are eliminated by using dot notation (similar to the notation used for objects in object-oriented languages like Java or Python). For examples, the instance in Figure 2 would be flattened as: { Successful: "False", Component1.type: "UFO", Component1.angle: 0.0, Component1.left: 0.1, Component1.right: 2.8, $\cdots$, "(On Component1 Component2)": "True", "(On Component2 Component3)": "True" }. The flattening process effectively eliminates the component and relational attributes while preserving their information in a form that can be used during partial matching to rename later instances; i.e., flat representations can be converted back into a form that contains component and relational attributes. Once converted into a flat representation the instances only contain nominal and numeric attributes that can be handled by existing incremental categorization approaches.

3.4 Categorization

To perform categorization on flattened instances the TRESTLE algorithm employs the COBWEB/3 algorithm (Fisher, 1987; McKusick & Thompson, 1990). This approach operates by recursively sorting instances into a hierarchical categorization tree. At each concept encountered during sorting, COBWEB/3 considers four possible actions to incorporate the instance into its tree: **creating** a new child concept to store the instance; **adding** the instance to the most similar child concept; **merging** the two most similar child concepts and then adding the instance to the resulting concept; and **splitting** the most similar child concept, promoting its children to be children of the current node, and recursing. To determine which action to take COBWEB/3 simulates each action and computes the category utility (Fisher, 1987) of the resulting child concepts. Like the similarity value being maximized during partial matching, category utility represents the increase in the average number of expected correct guesses achieved in the children compared to their parent concept, this measure is similar to the information gain heuristic used in decision tree learning (Quinlan, 1986). Mathematically, the category utility of a set of children $\{C_1, C_2, \cdots, C_n\}$ is:

$$CU(\{C_1, C_2, \cdots, C_n\}) = \frac{\sum_{k=1}^{n} P(C_k)[\sum_i \sum_j P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2]}{n}$$

where $P(C_k)$ is the probability of a particular concept given its parent, $P(A_i = V_{ij}|C_k)$ is the probability of attribute A_i having value V_{ij} in the child concept C_k, $P(A_i = V_{ij})$ is the probability of attribute A_i having value V_{ij} in the parent concept, and n is the number of child concepts. Each of these terms can be efficiently computed via a lookup of the probability tables stored in the parent and child concepts.

In order to handle numeric attributes, COBWEB/3 uses a normal probability density function to represent the probability of different values of a numeric attribute. Given this assumption, the sum of squared attribute-value probabilities is replaced with an integral of the squared probability density function, which in the case of a normal distribution is simply the square of the distribution's normalizing constant. Thus, $\sum_i \sum_j P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2$ is replaced with $\sum_i \frac{1}{\sigma_{ik}} - \frac{1}{\sigma_i}$ in the case of numeric attributes, where σ_{ik} is the standard deviation of values for the attribute A_i in child concept C_k and σ_i is the standard deviation of values for the attribute A_i in the

parent concept. For more detailed justification of this modification to category utility see (McKusick & Thompson, 1990).

3.5 Prediction and Clustering

During learning COBWEB/3 uses this categorization technique to incorporate new instances into its hierarchical tree. The resulting tree can then be used to make predictions about novel instances or provide clusterings of the examples to varying depths of specificity.

When being used to make predictions TRESTLE follows its normal partial matching and flattening procedures but makes some modifications to the COBWEB/3 categorization process to prevent the prediction from modifying TRESTLE's existing knowledge base. Firstly, as an instances is categorized down the tree, concepts do not update their probability tables to reflect the instance. Additionally, only the **creating** and **adding** operations are considered at each concept. When categorization encounters a leaf or a situation where the **creating** operation has the highest category utility it returns the current concept. The resulting concept's probability table can then be used to make predictions about attribute values of the instance, i.e., each missing attribute is predicted to have its most likely value according to the probability table.

In addition to prediction, TRESTLE can be used for clustering data by categorizing instances into the concept tree and assigning labels based on the resulting concepts. Given that TRESTLE represents concepts in a hierarchy, the resulting labels can be considered at varying levels of specificity. In the case that flat clusterings are desired, instances can be labeled by the concepts they are sorted into. In most cases the direct children of the root concept can be used as appropriate cluster labels. If more specific labels are desired then COBWEB/3's splitting procedure can be progressively applied to the resulting concepts until a desired level of specificity is achieved.

4. Evaluation

To evaluate the TRESTLE algorithm we assessed its performance on two tasks: incremental prediction of one missing attribute and unsupervised clustering. For each task, we compared the performance of TRESTLE to CFE, a previous approach for learning in structured domains. Beyond comparing machine-based approaches, we had humans perform both tasks and compared their performance to TRESTLE and CFE in order to assess which approach better models human learning.

Each of our evaluations make use of log data from *RumbleBlocks* players. Both the prediction and clustering tasks use records of final solutions to in-game levels. These records contain the position and orientation information of every tower block as well as a label of whether the tower stood or fell during the earthquake.

4.1 Prediction Task

For the prediction task we had each learning system (both machine approaches and humans) incrementally predict the success label of an example given their prior experience. In this incremental prediction task, learners were sequentially presented with *RumbleBlocks* towers and asked to predict if the tower successfully withstood the earthquake. They were then given correctness feedback on

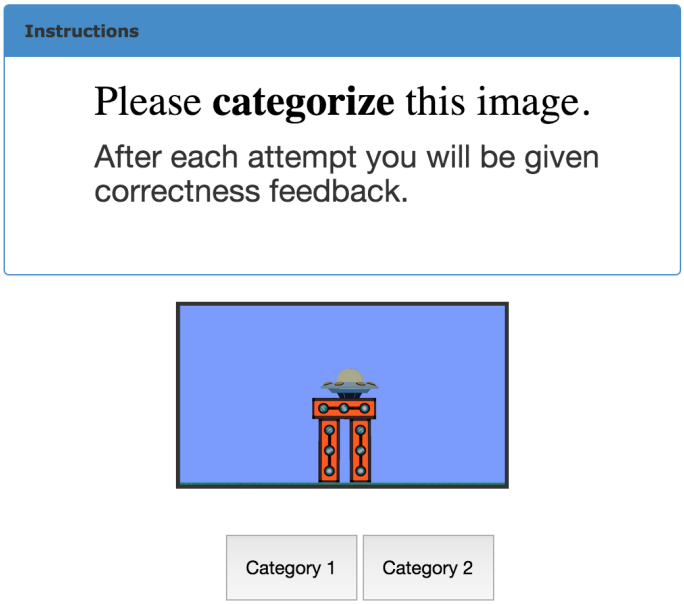


Figure 3: The incremental prediction task as presented to Amazon Mechanical Turk workers. Workers were asked to categorize 30 *RumbleBlocks* solutions into one of two categories: Category 1 (successful) or Category 2 (unsuccessful). They were not given any information about the meaning of the category labels, but they were given correctness feedback after each attempt.

their prediction (i.e., provided the success label). The instances for this task were taken randomly from all player solutions to the same level. To ensure that a naive strategy of base rate prediction would not work we chose a level whose overall success ratio was roughly even (a symmetry level in which 48.9% of player towers stood). Each learner was presented with 30 examples and the average accuracy performance for each approach was averaged across opportunities.

To get a sense of human learner performance at this task we used Amazon Mechanical Turk to have 20 humans perform incremental prediction using the interface shown in Figure 3. The humans were asked to decide if a screenshot of an example tower fit into "Category 1" (i.e., successful) or "Category 2" (i.e., unsuccessful) before being told which category the example belonged to. Category labels were presented abstractly to avoid human raters using their intuitive physics knowledge, which would not be accessible to either of the algorithmic approaches.

The CFE approach's performance at prediction was evaluated using a decision tree, specifically CART as implemented by the Scikit-learn library (Pedregosa et al., 2011), to incrementally predict the success labeling of solutions when provided the feature vector representation of each solution. In order to use decision trees for this incremental task, the tree was retrained after being presented with each new instance. The correctness of predictions at each opportunity were averaged across 1000 runs. We chose to use decision tree learning because it is prototypical of approaches used for learning logical rules and we are interested in exploring how TRESTLE might be used in similar capacities.

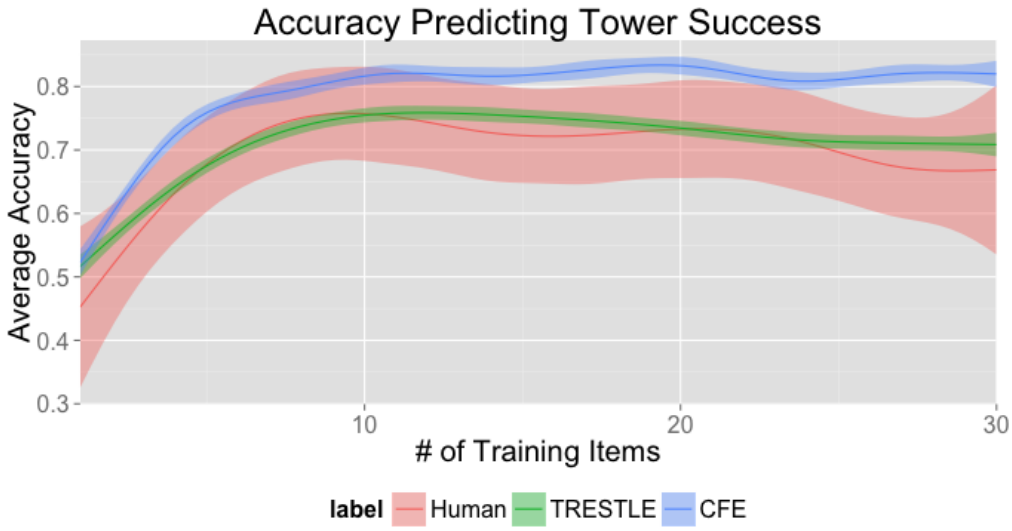


Figure 4: The average performance of 20 Humans, 1000 runs of TRESTLE, and 1000 runs of CFE for predicting whether solutions to a symmetry level survived an earthquake (shaded regions denote the 95% confidence intervals). We see that the incremental TRESTLE algorithm performs less well than the non-incremental CFE algorithm, but the TRESTLE performance more closely approximates human performance.

TRESTLE was applied to this task by having each instance categorized in a non-modifying way with its success label removed, the probability table of the returned concept was then used to predict the success of the instance by selecting whichever label had the highest probability within the concept. The instance was then categorized in the normal (i.e., modifying) fashion using all of its original information. Similar to CFE, the correctness of predictions at each opportunity was averaged across 1000 runs.

4.2 Prediction Results

The results of average prediction accuracy for 30 sequentially presented instances can be seen in Figure 4. Human labelers appear to converge around 75% accuracy suggesting that learning the success concept is somewhat difficult for humans to do, likely due to both the abstraction presented in the interface and the noise in the outcome variable. In contrast, the CFE approach performs better than humans, converging to roughly 85% accuracy. TRESTLE, on the other hand, performs roughly equal to the humans, with differences in their confidence intervals due primarily to having only 20 human raters.

CFE's increased learning rate (i.e., the steeper slope on the learning curve) is likely due to the fact that CFE is using a previously generated exhaustive grammar and that it is processing the information at each opportunity in a batch fashion, making it less susceptible to ordering effects.

Furthermore, its overall higher accuracy is due to the fact that it is optimizing for prediction of a single attribute (i.e., success). In contrast, TRESTLE learns incrementally from the examples, so it has a shallower learning rate, and it is optimizing for its ability to predict all attributes, so its overall accuracy predicting the single attribute of success is lower. Our results suggest that while these differences decrease TRESTLE's predictive accuracy they result in a model that more closely aligns with human performance. Thus, we argue that while CFE functions better as a pure machine learning system, achieving higher accuracy, it is a poorer model of human learners than TRESTLE.

4.3 Clustering Task

While the results of the incremental prediction task show that TRESTLE performs similar to humans, we are also interested in understanding if TRESTLE's underlying representation of knowledge is qualitatively similar to humans. A strong similarity between the TRESTLE and human organization of knowledge would strengthen our claim that TRESTLE is a better model of human learning than CFE. In particular, if TRESTLE's organization of concepts agrees with humans' organization, then it suggests TRESTLE's agreement with human data on the prediction task is not just a spurious result (i.e., TRESTLE is a worse learner than CFE at a task that humans also happened to do poorly on), but instead is the result of a similar organization of knowledge that produces worse prediction performance than CFE. To examine this aspect of TRESTLE we use a clustering task. For this evaluation we chose records from 3 levels of *RumbleBlocks* that were used as part of an in-game pre-posttest (one for each concept). These levels were attractive because they have a large number of player solutions and because the energy ball mechanic was removed for the pre-posttest setting allowing for a wider variance of player solutions than other levels.

To establish a human baseline we had 2 researchers from our lab act as human raters and independently hand cluster screenshots of all the player solutions to each level. The raters were told to group solutions that looked similar and to ignore any blocks that did not appear to be part of the solution tower (e.g., blocks off to the side of the frame) but were not given any other guidance as to how many groups to make on each level. To get a sense of the agreement between raters we calculated the Adjusted Rand Index (ARI) between their clusterings (Rand, 1971). The ARI measure is a generalization of Cohen's Kappa that allows for raters to use different numbers of categories. This value ranges from -1 to 1, with random clusterings producing an average ARI of 0. The average ARI between human raters across all three levels was 0.88, which is a high agreement. We chose one of the human clusterings to serve as a ground truth labeling of solutions when comparing to the machine approaches. In this labeling the average number of clusters per level was 33.

To create a comparison, we applied the CFE process to generate feature vector for the *RumbleBlocks* solutions. Then for each level, we clustered the feature vectors using the G-means algorithm (Hamerly & Elkan, 2004). G-means functions like a wrapper around k-means using a heuristic that tries to form clusters with a gaussian distribution among features to choose an optimal value for k. This process produced a clustering of solutions for each level. We performed this process 10 times and report the average and standard deviation in ARI between the CFE and human clusterings.

To examine TRESTLE's ability to cluster *RumbleBlocks* solutions, we had the algorithm create a categorization tree for each level. For each level, instances were categorized into the tree in a random order. Once all instances had been categorized once, we shuffled the instances and categorized them

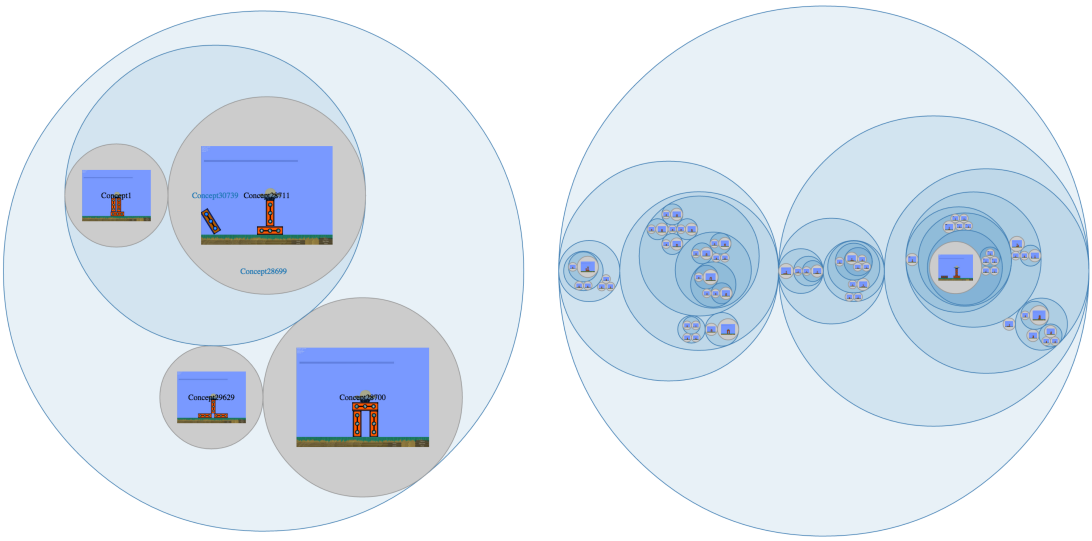


Figure 5: Two examples of hierarchical clusterings produced by the TRESTLE algorithm on a symmetry level. The clustering on the left (a) shows an early categorization tree after seeing six examples; the grey circles denote specific instances (or groups of identical instances) and the blue circles denote concepts that contain other concepts and instances. The clustering on the right (b) shows the same tree after incorporating 141 examples.

a second time, taking the resulting concepts they were sorted into as their labeling. While it is possible to cluster instances using a single run, we categorized the instances twice in order to give TRESTLE the opportunity to form an initial categorization tree before deciding on labels.

Both the human labeling and the CFE clustering produced flat clusterings, so for comparison purposes we transformed TRESTLE's hierarchical clustering into a non-hierarchical clustering. As mentioned earlier, a flat clustering of the examples can be produced by returning the most general unsplit concept as a label for each instance after performing a user-specified number of split operations to the root concept. For example, 0 splits will return the root concept as the label for every instance, 1 split will return the children of the root as concept labels, and 2 splits will return a more refined set of concept labels produced by further splitting one of the children. This process is similar to the one used to produce flat clusterings from agglomerative clustering techniques. Arguably the hierarchical clustering is more informative, for an example of a tree see Figure 5, but we flattened the clusterings purely for the purposes of evaluating the approach with non-hierarchical human clusterings. To provide a sense for how the clusterings at different levels of the hierarchy agree with human clusterings, we report the clustering results for the first three splits of the categorization tree. Similar to the CFE evaluation, we performed this process 10 times and reprort the average and standard deviaiton in ARI between the human labelings and each split of the TRESTLE labelings.

Table 1: The agreement (ARI) between the model clusterings and human clusterings. We report the average and STD of this measure across 10 clustering runs.

Level	Model	ARI	STD
Center of Mass (n=251)	Trestle (1 split)	0.37	0.03
	Trestle (2 splits)	0.50	0.08
	Trestle (3 splits)	**0.54**	0.13
	CFE	0.51	0.08
Symmetry (n=249)	Trestle (1 split)	0.16	0.08
	Trestle (2 splits)	0.34	0.08
	Trestle (3 splits)	0.44	0.08
	CFE	**0.47**	0.04
Wide Base (n=254)	Trestle (1 split)	**0.56**	0.02
	Trestle (2 splits)	0.47	0.10
	Trestle (3 splits)	0.41	0.05
	CFE	0.42	0.02

4.4 Clustering Results

The ARI results between the two machine clusterings and human clustering can be seen in Table 1. For the Center of Mass and Wide Base levels, the TRESTLE algorithm produces clusterings that have the highest agreement with the human clusterings. In the Symmetry level, the CFE approach produces the most human-like clusterings. In general, TRESTLE seems to have better performance that CFE, despite its incremental nature.

The one level where CFE outperforms TRESTLE (i.e., the Symmetry level) might be due to a high number of human clusters on this level (Center of Mass had 33 clusters, Symmetry had 40 clusters, and Wide Base had 24 clusters). The Wide Base level has the least number of clusters (as determined by the humans), which is likely why only a single split in TRESTLE produced the highest agreement. In contrast, the Center of Mass and Symmetry levels had higher numbers of clusters and agreement between TRESTLE and humans increased as the number of splits was increased. For the Symmetry level three splits of the TRESTLE categorization tree produced almost the same agreement as the CFE algorithm; however, because the symmetry level had the highest number of human clusters it is likely that more splits would have improved TRESTLE's agreement with the human clusterings. Overall, TRESTLE was capable of producing non-hierarchical clusterings that had good agreement with human clusterings; however, TRESTLE is also able to return hierarchical categorizations that arguably contain more information. For the purposes of better understanding the space of different solutions to a *RumbleBlocks* level they are potentially more interpretable; for example, see the tree visualization in Figure 5.

5. Discussion

In this paper we have presented the TRESTLE algorithm and compared its performance on prediction and clustering tasks with humans and with CFE, an existing approach for feature extraction in structured domains. For the prediction task, we found that CFE achieved higher accuracy more quickly than TRESTLE due to its ability to process information in batches and to focus on predicting a single attribute value (i.e., success). However, TRESTLE's slower rate of learning and lower overall accuracy agreed more closely with human performance. This is likely due to the fact that both TRESTLE and the humans are learning incrementally and are constructing concepts that can be used to predict any missing attribute value (not just success). Another important difference between CFE and TRESTLE is that the CFE approach required all of the examples up front (without their success labels), in order to learn a set of grammar rules that would work for all of the examples. Alternatively, the grammar rules could have been regenerated for the examples at each step in the prediction task. TRESTLE, on the other hand, used a more flexible partial matching and incremental learning approach that allowed it to update its knowledge online and make predictions given novel structures. These capabilities make TRESTLE a more flexible learning approach that, like humans, can quickly adapt and learn from new situations. Thus, while CFE has better overall prediction performance, we argue that TRESTLE is a better model of human learning.

For the clustering task, we found that different splits of TRESTLE's categorization tree tended to have higher agreement with human clusterings than CFE, a more specialized non-hierarchical clustering approach. However, TRESTLE's category utility heuristic tended to favor a smaller number of child concepts for each concept in the tree. Thus, for levels that had a higher number of solution clusters (as determined by the human coders) more splits of the TRESTLE tree tended to have better agreement with the human clusterings than less splits. While we did not explore computational approaches to determining the optimal number of splits for producing flat clusterings, we believe our results demonstrate that the organization of TRESTLE's conceptual knowledge has a reasonably high agreement with humans.

One general interpretation of our results is that TRESTLE is a better model of human performance while CFE is a better machine learning approach (i.e., achieves greater performance). Both our prediction and clustering results support this interpretation. First, TRESTLE's predictive accuracy is closer to human predictive accuracy, while CFE has higher predictive accuracy than both TRESTLE and humans. Further, TRESTLE's organization of knowledge more closely agrees with humans than the organization produced using CFE. The two approaches differs, in part, because of the different motivations underlying their designs. CFE was originally created as a means of mapping instances in a structured domain into feature vectors so that other learning algorithms (e.g., decision trees) could function in the space. It was designed primarily for use in offline settings, e.g., post-hoc analysis of *RumbleBlocks* solutions, rather than for making online decisions. This is why the algorithm makes use of exhaustive grammar generation and computes features based on all possible parses of structures, in an attempt to maximize its coverage of the space despite the additional training costs. TRESTLE on the other hand arises out of existing theory on human categorization and takes a more probabilistic approach to the problem of coverage. Rather than focus on multiple potential interpretations of a tower it looks at probabilistic similarity with existing knowledge. Additionally it maintains more cognitive plausibility by avoiding exhaustive processes, e.g. it does not

seem very plausible that new players of *RumbleBlocks* have a pre-existing *RumbleBlocks* grammar, nor that they are visually parsing towers using a large number of grammar rules (approx. 6000 were generated by CFE for *RumbleBlocks*).

Our results serve as an initial demonstration of TRESTLE's abilities to operate in domains with nominal, numeric, component, and relational attributes. While machine learning approaches exist that support each of these characteristics individually, rarely are they integrated into a single learning approach. Further, systems that do exist to support numeric, nominal, and relational attributes (e.g., FOIL) do not typically operate in an incremental fashion. Thus, we argue that TRESTLE is one of the few systems that supports all of these capabilities making it particularly well suited for modeling human categorization in a wide range of applications. For example, there is an active debate about whether blocking or interleaving similar types of problems is better for student learning (Carvalho & Goldstone, 2013). One hypothesis is that blocking problems supports students' ability to map structure across similar problems and thus supports the acquisition of structural knowledge. The contrasting hypothesis is that interleaving problems better supports students' ability to identify discriminating features. Previous work modeling human learning in intelligent tutors with blocked vs. interleaved problem types has suggested that interleaved instruction is preferable (Li, Cohen, & Koedinger, 2012). However, this previous work used a non-incremental approach and is given pre-trained structural knowledge (similar to CFE), so it is more likely to show a preference for interleaved instruction. In contrast, a model like TRESTLE would be uniquely qualified to explore the issue of blocked vs. interleaved instruction because it models the incremental acquisition of structural concepts.

Finally, our analysis demonstrates that the TRESTLE system is capable of performing both supervised and unsupervised learning tasks. We argue that this makes TRESTLE a more general model of learning than CFE, which had to employ separate machine learning algorithms (i.e., decision trees for prediction and k-means for clustering) for the two tasks. In the current work we evaluated the system on separate prediction and clustering tasks, but the TRESTLE algorithm should also support more hybridized tasks that share different amounts of supervision. In prior categorization work, COBWEB models have been used to incrementally predict multiple missing attribute values of instances, rather than just a single attribute value (Fisher, 1987). TRESTLE is also capable of predicting multiple missing attribute values, but we did not perform this kind of evaluation in the current work because it is unclear how to evaluate prediction performance with component attributes (e.g., should a system be evaluated in terms of its ability to predict entire components or different attribute values of components).

6. Future Work

We presented TRESTLE and demonstrated its ability to incrementally learn concepts from mixed-data environments in both a supervised and unsupervised fashion. However, our results are only a preliminary evaluation of the current system, and more work is necessary to fully assess its capabilities. In particular, we would like to evaluate TRESTLE's ability to perform more flexible prediction tasks. While it is unclear how best to assess flexible prediction in structured domains, we would like to more fully assess the system's generative capabilities. Thus, in future work we might

explore the system's ability to perform pattern completion in structured domains. In the domain of *RumbleBlocks*, this might consist of providing TRESTLE with incomplete towers and asking it to complete them.

In addition to assessing the current systems capabilities, we would like to explore how the current system can be more closely integrated with approaches for planning and executing action. For example, the system could be extended to support "functional" attributes. These attributes would have actions as values, allowing the system to decide on what action to take given an instance. Unlike systems that separate concept and skill knowledge, this extension would allow TRESTLE to use functional information to guide its formation of concepts and vice versa.

We would also like to explore the implications of TRESTLE for supporting designers. For example, prior work has looked at how grammar-based patterns can be learned from structured examples and how these patterns can be used to generate novel designs (Talton et al., 2012). However, current approaches do not support mixed-data types or missing attributes. TRESTLE could be used to extend these existing approaches to support these capabilities, allowing it to support novel tasks (e.g., completing partially specified designs). Additionally, we would like to explore how TRESTLE could be used to support game designers in better understanding the space of player solutions. This application of supporting designers was part of the original motivation beyond CFE, but our clustering results demonstrate that TRESTLE may be better suited to this task and even be able to provide more information through its hierarchical organization of concepts.

Finally, we would like to explore how TRESTLE can be used as a computational model of human learning in educational technologies. In particular, we would like to use TRESTLE to stimulate student's learning in various educational environments (e.g., in Intelligent Tutoring Systems or in Educational Games). In these contexts, the model could be used to test and improve different instructional materials before they are given to students (e.g., to determine the best ordering of levels in *RumbleBlocks*). Additionally, the TRESTLE system can be used to automatically construct knowledge models through training. These models, either of expert or buggy knowledge, can then be used in Intelligent Tutoring Systems to provide students with improved context-sensitive feedback and instruction.

7. Conclusion

In this work, we introduced TRESTLE, an algorithm for incremental concept formation in structured domains. We demonstrated the system's ability to perform prediction and clustering tasks in a domain with nominal, numeric, component, and relational information. Our results showed that TRESTLE can achieve performance that is comparable to CFE, a previously developed non-incremental approach. Further, our results suggest that while CFE can achieve higher predictive accuracy, TRESTLE better matches human performance. Our results demonstrate TRESTLE's capabilities and provide an initial demonstration of how it can be used to model human categorization.

Acknowledgements

We would like to thank the designers of *RumbleBlocks* who provided our data. Additionally, we would like to thank Victor Hwang, Kelly Rivers, and Douglas Fisher for discussing earlier versions

of this work. This work was supported in part by the Department of Education (#R305B090023), National Science Foundation (#SBE-0836012), and the DARPA ENGAGE research program under ONR Contract Number N00014-12-C-0284.

References

Carvalho, P. F., & Goldstone, R. L. (2013). Putting category learning in order: Category structure and temporal arrangement affect the benefit of interleaved over blocked study. *Memory & Cognition*, *42*, 481–495.

Christel, M. G., Stevens, S. M., Maher, B. S., Brice, S., Champer, M., Jayapalan, L., Chen, Q., Jin, J., Hausmann, D., Bastida, N., Zhang, X., Aleven, V., Koedinger, K. R., Chase, C., Harpstead, E., & Lomas, D. (2012). RumbleBlocks: Teaching science concepts to young children through a Unity game. *Proceedings of the Seventeenth International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games* (pp. 162–166).

Feigenbaum, E. A., & Simon, H. A. (1984). EPAM-like Models of Recognition and Learning. *Cognitive Science*, *8*, 305–336.

Fisher, D. H. (1987). Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, *2*, 139–172.

Fisher, D. H., & Yoo, J. (1993). Categorization, Concept Learning, and Problem Solving: A Unifying View. In G. Nakamura, R. Taraban, & D. Medin (Eds.), *The psychology of learning and motivation*, 219–255. San Diego, CA: Academic Press.

Forbus, K. D., Gentner, D., & Law, K. (1995). MAC/FAC: A model of similarity-based retrieval. *Cognitive Science*, *19*, 141–205.

Hamerly, G., & Elkan, C. (2004). Learning the k in k-means. *Proceedings of the Sixteenth Conference on Advances in Neural Information Processing Systems* (pp. 281–288).

Harpstead, E., MacLellan, C. J., Koedinger, K. R., Aleven, V., Dow, S. P., & Myers, B. A. (2013). Investigating the Solution Space of an Open-Ended Educational Game Using Conceptual Feature Extraction. *Proceedings of the Sixth International Conference on Educational Data Mining*.

Holyoak, K. (2005). Analogy. In K. Holyoak & R. G. Morrison (Eds.), *The cambridge handbook of thinking and reasoning*, 117–142. Cambridge University Press.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Langley, P., Laird, J. E., & Rogers, S. (2008). Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, *14*, 141–160.

Li, C., & Biswas, G. (2002). Unsupervised Learning with Mixed Numeric and Nominal Data. *IEEE Transactions on Knowledge and Data Engineering*, *14*, 673–690.

Li, N., Cohen, W. W., & Koedinger, K. R. (2012). Problem Order Implications for Learning Transfer. *Proceedings of the Eleventh International Conference on Intelligent Tutoring Systems* (pp. 185–194). Springer Berlin Heidelberg.

Li, N., Schreiber, A. J., Cohen, W. W., & Koedinger, K. R. (2012). Efficient Complex Skill Acquisition Through Representation Learning. *Advances in Cognitive Systems*, *2*, 149–166.

McKusick, K., & Thompson, K. (1990). *COBWEB/3* (Technical Report FIA-90-6-18-2). NASA Ames Research Center, Moffett Field, California.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Quinlan, J. R., & Cameron-Jones, R. M. (1995). Induction of logic programs: FOIL and related systems. *New Generation Computing*, *13*, 287–312.

Rand, W. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, *66*, 846–850.

Talton, J. O., Yang, L., Kumar, R., Lim, M., Goodman, N. D., & Mech, R. (2012). Learning Design Patterns with Bayesian Grammar Induction. *Proceedings of the Twenty-Fifth Annual ACM Symposium on User Interface Software and Technology* (pp. 1–11).

Thompson, K., & Langley, P. (1991). Concept formation in structured domains. In D. H. Fisher, M. J. Pazzani, & P. Langley (Eds.), *Concept formation: Knowledge and experience in unsupervised learning*, 127–161. San Mateo, CA: Morgan Kaufmann.

Todorovic, D. (2008). Gestalt principles. *Scholarpedia*, *3*, 5345.

Vanlehn, K., Jones, R. M., & Chi, M. T. H. (1992). A Model of the Self-Explanation Effect. *The Journal of Learning Sciences*, *2*, 1–59.

Zhu, X., Lee, Y., Simon, H. A., & Zhu, D. (1996). Cue recognition and cue elaboration in learning from examples. *Proceedings of the National Academy of Sciences*, *93*, 1346–1351.

Zhu, X., Rogers, T., Qian, R., & Kalish, C. (2007). Humans perform semi-supervised classification too. *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence* (pp. 864–869). Menlo Park, CA.