Advances in Cognitive Systems 4 (2016) 131-150 Submitted 8/2015; published 6/2016

# TRESTLE: A Model of Concept Formation in Structured Domains

**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

The literature on concept formation has demonstrated that humans are capable of learning concepts incrementally, with a variety of attribute types, and in both supervised and unsupervised settings. Many models of concept formation focus on a subset of these characteristics, but none account for all of them. In this paper, we present TRESTLE, an incremental account of probabilistic concept formation in structured domains that unifies prior concept learning models. 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 updates its knowledge by partially matching novel structures and sorting them into its categorization tree. Finally, the system supports mixed-data representations, including nominal, numeric, relational, and component attributes. We evaluate TRESTLE's performance on a supervised learning task and an unsupervised clustering task. For both tasks, we compare it to a nonincremental model and to human participants. We find that this new categorization model is competitive with the nonincremental approach and more closely approximates human behavior on both tasks. These results serve as an initial demonstration of TRESTLE's capabilities and show that, by taking key characteristics of human learning into account, it can better model behavior than approaches that ignore them.

## 1. Introduction

Humans can improve their performance with experience. To better understand these capabilities, numerous research efforts have constructed computational models of human learning (Vanlehn, Jones, & Chi, 1992; Fisher & Langley, 1990; Li, Schreiber, Cohen, & Koedinger, 2012c; Laird, Rosenbloom, & Newell, 1986; Langley, Laird, & Rogers, 2009b). 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 & Langley, 1990). In contrast, most recent work in cognitive architectures emphasizes the generation of solutions to problems or the execution of actions (Langley et al., 2009b). 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, Lee, Simon, & Zhu, 1996). Thus, we argue that more research should be conducted on 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 (Giraud-Carrier, 2000; Love, Medin, & Gureckis, 2004). People revise learned concepts given new information rather than reconsidering all prior experiences and learning completely new structures. This lets them improve their performance given more experience and to flexibly adapt their knowledge and understanding in response to novel situations in an efficient way. This approach contrasts with the main thrust of research in machine learning, which emphasizes nonincremental approaches in an effort to aid performance. Thus, while incremental learners can 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 fractions problems, the order in which they receive problems affects their learning (Rau, Aleven, & Rummel, 2013). More generally, how best to order practice problems for humans in order to promote robust learning remains an open research question (Li, Cohen, & Koedinger, 2012b; Carvalho & Goldstone, 2013) that would stand to benefit from computational models of incremental acquisition.

A second characteristic of human categorization is that it occurs in a wide range of environments that can involve structural, relational, nominal, and numeric information. For example, the work of Carvalho and Goldstone (2013) has shown that the structural similarity of objects present in examples affects human concept formation. Other work 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 (Gennari, Langley, & Fisher, 1989; Li & Biswas, 2002; Quinlan, 1986) information, less emphasis has been placed on learning with structural content (Thompson & Langley, 1991; Forbus et al., 1995). Further, few studies exist that explore how best to model the integration of multiple data types. Of the existing methods, only a subset integrate nominal and numeric data (Li & Biswas, 2002; Quinlan, 1986; Gennari et al., 1989). Finally, even fewer studies combine relational (Quinlan & Cameron-Jones, 1995) or structural (Thompson & Langley, 1991; Forbus et al., 1995) information. We claim that any system attempting to model human categorization should integrate all of these data types into a single account.

A third characteristic of human concept learning is that it occurs in both supervised and unsupervised settings (Love et al., 2004). In many cases, these two modes of learning are treated as distinct, but there is evidence that humans can improve their performance on unsupervised tasks given supervision on a different task (Zhu, Rogers, Qian, & Kalish, 2007). Others have found the reverse, that humans can improve their performance on supervised tasks given experience on other unsupervised tasks (Kellman & Garrigan, 2009). Taken together, these findings suggest that humans share knowledge across these settings, and that models of human categorization should be capable of operating with and without supervision, with knowledge used for one influencing the other.

To explore these aspects of human categorization, we developed TRESTLE, an incremental model of probabilistic concept formation in structured domains that builds on prior research in categorization (Fisher, 1987; McKusick & Thompson, 1990; Thompson & Langley, 1991). This approach maps novel structures to its existing knowledge in an online fashion and updates its categorization knowledge based on these new structures. TRESTLE also handles mixed representations, letting it function with nominal, numeric, relational, and component attributes. Finally, the approach

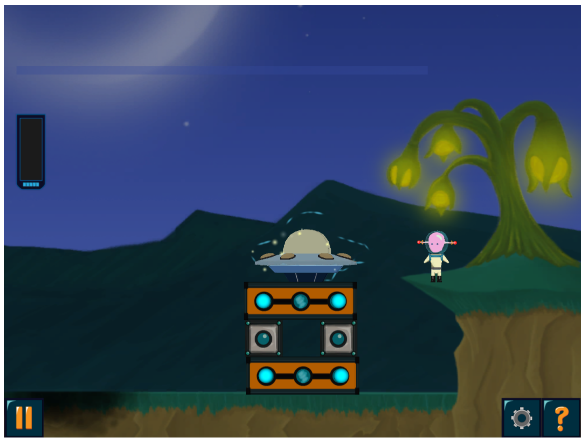


*Figure 1.* A screenshot of a player building a tower in *RumbleBlocks*. The final tower must cover the light blue energy balls and survive a simulated earthquake to be successful.

supports partial matching, so it can process incomplete or partially specified input. These features let TRESTLE use its categorization knowledge for predicting missing attributes in supervised settings and for clustering examples in purely unsupervised situations. In this paper, we present an example domain based on the educational game *RumbleBlocks* (Christel et al., 2012) that contains nominal, numeric, relational, and structured information, describe TRESTLE's approach to learning concepts in this domain, and present a preliminary evaluation of the algorithm that compares it with more specialized approaches and with humans. We show that the model can learn probabilistic concepts given supervision and make predictions based on these concepts at human levels of performance. Additionally, for unsupervised clustering, it produces clusters that have reasonably high agreement with human clusterings. These results provide an initial demonstration of how TRESTLE models human concept formation in structured domains.

## 2. The *RumbleBlocks* Domain

To investigate human learning in a rich domain, we have explored data from *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. Players build a tower out of blocks in a two-dimensional, continuous world with a realistic rigid body physics simulation, designing 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. After 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 he fails and must try the level again.

Each level of the game is designed to emphasize one of three primary concepts of structural stability and balance: objects that are symmetrical, that have wide bases, and that have lower centers of mass are more stable. While the drag-and-drop actions of tower construction are necessary to succeed in the game, they are not relevant its educational goals. 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 several interesting hurdles when attempting to model players' conceptual learning. The biggest challenge is in representing the richly structured space of the game. States exist in a continuous space but also possess nominal (e.g., block types) and structural information (e.g., high level patterns like arches). Additionally, the physics simulation that models the earthquake is nondeterministic and susceptible to minor perturbations in rigid body physics. Therefore, two towers that appear similar have a small chance of being treated differently in terms of the success criteria, resulting in noisy evaluation feedback. This noisy feedback makes it more challenging to learn the correct concepts and makes ordering effects more pronounced; if a borderline tower fails early in a student's problem sequence, then he will be more likely to avoid similar structures throughout his play than if the tower had succeeded. Finally, *RumbleBlocks* lets students partake in both supervised and unsupervised learning. They receive supervised feedback regarding the success of their towers, but are left to learn the game's target concepts (i.e., wide base, low center of mass, and symmetry) in an unsupervised way.

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 CFE or Conceptual Feature Extraction (Harpstead, MacLellan, Koedinger, Aleven, Dow, & Myers, 2013). CFE, which is similar to the grammar learning approach used by SimStudent (Li, Cohen, & Koedinger, 2012a), 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. 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 are then used by traditional learning methods for prediction or clustering. The goal was to construct features that capture the maximal amount of structural information in the examples so that this information could be used in learning.

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), but it has not been evaluated as part of a model of human categorization in *RumbleBlocks*. There are several obstacles to using CFE to model human categorization. First, the algorithm runs in a batch fashion instead of incrementally and requires all the examples for each task to generate its grammar. Thus, any examples with previously unseen features would require the learning of an entirely new grammar. Second, the grammar rule formalism used by CFE breaks down in continuous environments and when two-dimensional objects cannot be cleanly decomposed across axes. For example, when given a set of blocks in a spiral pattern, it cannot decompose the structure along the horizontal or vertical axes; subsequently, it cannot reduce the structure to the primitive grammar elements and fails at parsing the block structure. Finally, CFE and other similar grammar approaches (Talton, Yang, Kumar, Lim, Goodman, & Mech, 2012) cannot handle missing data in that they cannot partially match during parsing. Thus, if an attribute is missing from the training set, then examples containing that attribute cannot be parsed during run time. Similarly, if an attribute is always present in training but not at run time, then some examples would not be parsable.

## 3. The TRESTLE Model

In order to better model human concept formation in the *RumbleBlocks* domain we developed TRESTLE,[1] a system that incrementally constructs a categorization tree given the sequential presentation of instances. This categorization tree can then be used to make predictions about the given instances or to generate cluster labelings. At a high level, TRESTLE 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; and
3. **Categorization**, which incorporates the example into existing knowledge and, if necessary, makes predictions.

In this section we describe how TRESTLE represents both instances and concepts. We also clarify the processing steps that it carries out to learn from each instance.

### 3.1 Instance and Concept Representations

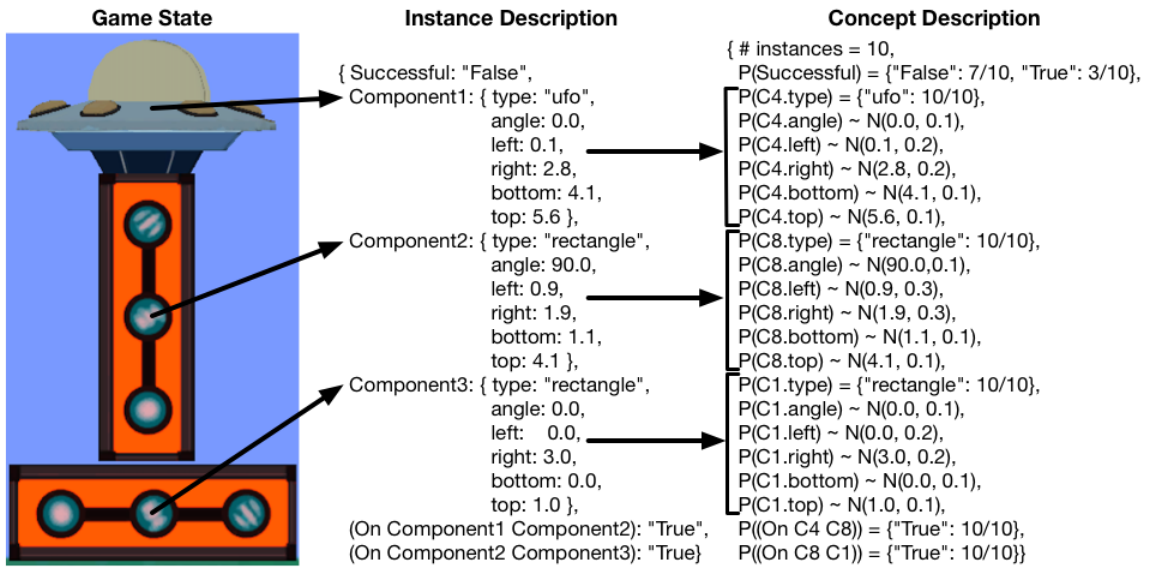
TRESTLE uses two representations to support learning: one for instances (i.e., specific examples it encounters) and one for concepts (i.e., generalizations of examples in memory). Instances are encoded in TRESTLE as sets of attribute values, where each attribute can have one of four types: **nominal** (e.g., the type or color of a block); **numeric** (e.g., the position of a block in continuous space); **component** (e.g., a block with type and dimension attributes is a component of a greater tower); and **relational** (e.g., that two blocks are vertically adjacent). 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 encoded, but in practice the component information is often sufficient for learning meaningful concepts. Our subsequent evaluation does not use these hand-generated relations; we only used the component and success information that was automatically provided by the game engine.

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 tracks how often each attribute value occurs in the underlying instances (e.g., see Figure 2). These tables can be used to find 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 searches for the best partial match between the instance and the root concept, which contains the attribute-value counts of all previous instances.

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



```
{ 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}
```

```
{ # instances = 10,
  P(Successful) = {"False": 7/10, "True": 3/10},
  P(C4.type) = {"ufo": 10/10},
  P(C4.angle) ~ N(0.0, 0.1),
  P(C4.left) ~ N(0.1, 0.2),
  P(C4.right) ~ N(2.8, 0.2),
  P(C4.bottom) ~ N(4.1, 0.1),
  P(C4.top) ~ N(5.6, 0.1),
  P(C8.type) = {"rectangle": 10/10},
  P(C8.angle) ~ N(90.0,0.1),
  P(C8.left) ~ N(0.9, 0.3),
  P(C8.right) ~ N(1.9, 0.3),
  P(C8.bottom) ~ N(1.1, 0.1),
  P(C8.top) ~ N(4.1, 0.1),
  P(C1.type) = {"rectangle": 10/10},
  P(C1.angle) ~ N(0.0, 0.1),
  P(C1.left) ~ N(0.0, 0.2),
  P(C1.right) ~ N(3.0, 0.2),
  P(C1.bottom) ~ N(0.0, 0.1),
  P(C1.top) ~ N(1.0, 0.1),
  P((On C4 C8)) = {"True": 10/10},
  P((On C8 C1)) = {"True": 10/10}}
```

*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. The arrows denote the mapping between blocks, instance components, and components in the concept.

Each match consists of a partial mapping between the component attributes in the instance to component attributes in the root concept. For example, Component1 in the instance shown in Figure 2 might be renamed to C4. When component attributes are renamed, any relation attributes that reference them are also updated; for instance (On Component1 Component2) might become (On C4 C8). TRESTLE currently only performs matching at the root for efficiency reasons, although future work might explore effects of matching on intermediate concepts.

The quality of a match is determined by how similar the resulting instance is to the root concept. Similarity is computed as the expected number of attribute values that can be correctly guessed after incorporating the matched instance into the root. 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 best-first search. 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. By default, TRESTLE uses this heuristic to guide a beam search (Wilt, Thayer, & Ruml, 2010) under the assumption that greedy approaches are more psychologically plausible than optimal ones. However, this heuristic is admissible and ensures that the best possible match is found when used with the A* algorithm, which TRESTLE optionally supports. The mapping between the instance and concept in Figure 2 is an example of a best match using beam search.

### 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, component attributes are eliminated by concatenating component and attribute names into a single attribute name. Throughout the paper we denote this conventionally using dot notation. 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. 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 approaches to incremental categorization.

### 3.4 Categorization

To categorize flattened instances, TRESTLE employs the COBWEB algorithm (Fisher, 1987; McKusick & Thompson, 1990), which recursively sorts instances into a hierarchical categorization tree. At each concept encountered during sorting, it considers four possible operations (shown in Figure 3) to incorporate the instance into its tree: **adding** the instance to the most similar child concept; **creating** a new child concept to store the instance; **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 concept, and recursing. COBWEB determines which operation to perform by simulating each action and computing the category utility (Fisher, 1987) of the resulting child concepts. Like the similarity value being maximized during partial matching, this represents the increase in the average number of expected correct guesses achieved in the children compared to their parent concept; thus it is similar to the information gain heuristic used in decision-tree induction (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.

For numeric attributes, COBWEB uses a normal probability density function to encode the probability of different values of a numeric attribute. Given this assumption, the sum of squared

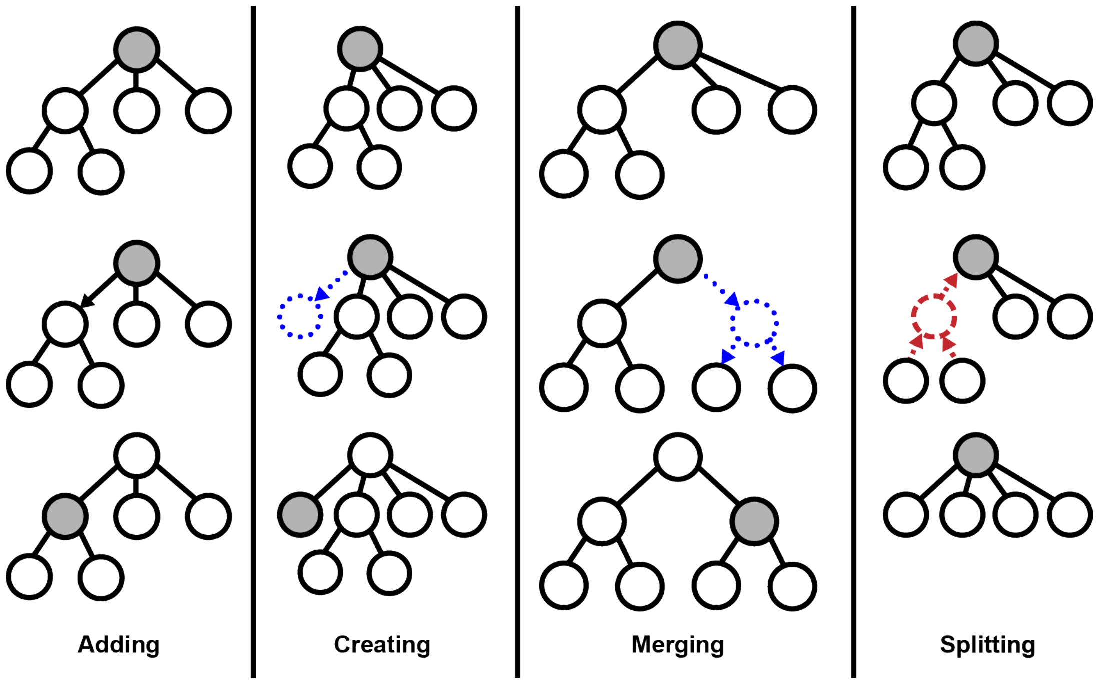
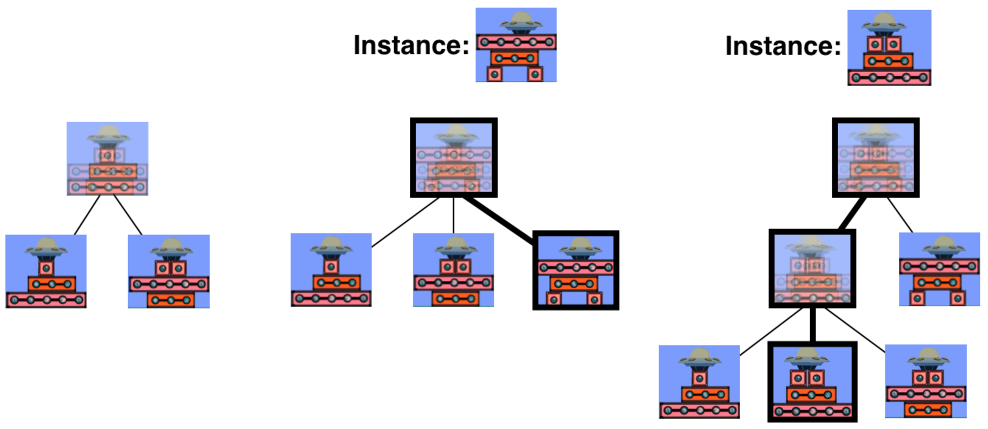


*Figure 3.* The four operations used by the COBWEB (Fisher, 1987) to incorporate matched instances into its categorization tree. Each shaded node depicts the location of the instance being sorted into the tree before and after an operation. The blue dotted lines represent nodes and links that are being added to the tree and red dashed lines represent nodes and links that are being removed from the tree.

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 $\sigma_{ik}$ is the standard deviation of values for the attribute $A_i$ in child concept $C_k$ and $\sigma_i$ 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 Gennari et al. (1989).

Figure 4 shows a simple example of a TRESTLE categorization tree from the *RumbleBlocks* domain and how it is updated in response to new instances. For the purposes of the example, we represent concepts as the images of the instances they contain rather than probability tables. Thus, concepts with a single instance are a single image while concepts with many instances are represented as multiple overlapping images to reflect their probabilistic nature. Figure 4(a) shows an existing categorization tree containing two previously incorporated instances. In Figure 4(b), TRESTLE is presented with a new instance, evaluates the category utility of its four operations, and decides that **creating** a concept to represent the new instance has the highest utility. The new concept is shown in the right leaf of Figure 4(b). In Figure 4(c), TRESTLE is presented with another example. In this case, the new instance is similar to two existing concepts, so TRESTLE decides that merging the two concepts and adding the new case to the merged concept has the highest category utility. After performing the **merge** action, the system continues categorization by considering the



(a) Original tree (b) Creating a new concept (c) Merging two existing concepts and creating a new one

*Figure 4.* A simple example of how TRESTLE's categorization tree is updated in response to two new instances. The original tree (a) is modified in (b) and (c) to incorporate the instances shown at the top. In each case, the path of the instance through the categorization tree is shown in bold. The concepts are depicted as overlapping images of the instances that they contain to represent their probabilistic nature.

four operations at the merged node and decides that **creating** a new concept to represent the instance has the highest utility. The newly created concept is depicted by the middle node at the bottom of the tree in Figure 4(c). When TRESTLE encounters future instance, it updates the tree in a similar fashion using the four operations shown in Figure 3.

## 3.5 Prediction and Clustering

During learning, COBWEB uses this categorization technique to incorporate new instances into its conceptual hierarchy. 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 TRESTLE makes predictions, it follows its normal partial matching and flattening procedures but employs COBWEB's non-modifying categorization process. In this version of categorization, an instance is sorted down the tree, but concepts do not update their probability tables to reflect the example. Additionally, the system only considers the **creating** and **adding** operations 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 is then used to make predictions about attribute values of the instance, with each missing attribute predicted to have its most likely value according to the table.

In addition to prediction, TRESTLE can cluster data by using COBWEB to categorize instances into its concept tree and assign labels based on the selected concepts. Using this process, TRESTLE produces a hierarchical clustering of the data. Alternatively, TRESTLE can convert this hierarchical clustering into a flat clustering by labeling all instance by the most general unsplit label (initially the root concept for all instances). For more specific cluster labels, TRESTLE progressively applies

COBWEB's **splitting** procedure to the most general unsplit concept labels until a desired level of specificity is achieved.

## 4. Experimental Evaluation

We designed TRESTLE to model three key aspects of human concept learning: incremental processing, use of multiple information types (nominal, numeric, component, and relational), and operation in both supervised and unsupervised settings. Further, implicit in this design is the claim that, by taking these three aspects of human categorization into account, we can better model human categorization. Thus, we had two goals for our experimental evaluation: to demonstrate TRESTLE's abilities to perform incremental learning with multiple types of information in both supervised and unsupervised capacities, and to show that, by taking these into account, it models human performance better than similar systems that do not.

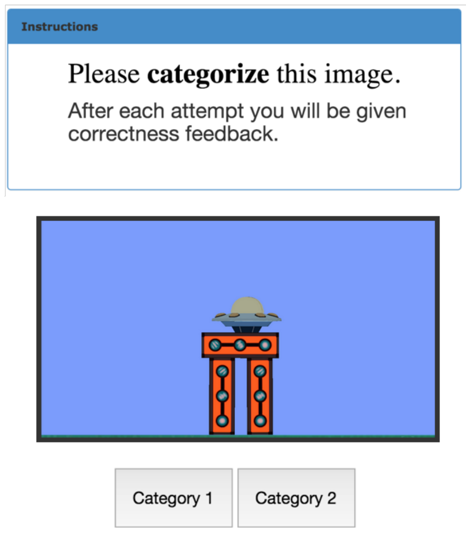
To demonstrate TRESTLE's functionality, we assessed its behavior on two tasks: a supervised learning task and an unsupervised clustering task. For each task we used player log data from the *RumbleBlocks* domain, which contains nominal, numeric, and component information. Additionally, we wanted to test our claim that the system models human categorization better than alternative ones. To achieve this goal, we compared TRESTLE to CFE, a nonincremental approach to learning in structured domains. To establish a baseline for the comparison, we had humans perform both tasks. We then compared the behavior of both systems to the human behavior to assess which approach better models the latter's learning.

### 4.1 Task 1: Supervised Learning

For the supervised learning task, each learner (TRESTLE, CFE, and human) was sequentially presented with *RumbleBlocks* towers and asked to predict if the tower successfully withstood the earthquake. They were then given correctness feedback about their prediction (i.e., provided the success label). Instances for this task were taken randomly from all player solutions to the same level of *RumbleBlocks*. To avoid a naive strategy of base rate prediction, we chose a level whose overall success ratio was roughly even (a symmetry level in which 48.9% of player towers stood). We presented each learner with 30 randomly selected and randomly ordered examples and averaged the accuracy for each approach across opportunities.

To determine human behavior for this task, we used Amazon Mechanical Turk to have 20 humans perform sequential prediction using the interface shown in Figure 5. We asked participants to decide if a screenshot of an example tower fit into "Category 1" (successful) or "Category 2" (unsuccessful) before telling them the correct category. We presented labels abstractly to avoid human raters using their intuitive physics knowledge, which would not be accessible to either of the computational models.

CFE is a nonincremental approach for handling structural information that must be paired with another learning algorithm in order to do prediction or clustering. To sequentially predict the success labeling of solutions, we paired CFE with CART, a decision-tree learner implemented in the Scikit-learn library (Pedregosa et al., 2011). Because CART is nonincremental, we rebuilt its decision tree after viewing each new training instance. We applied this approach to 1,000 random training


Instructions

Please **categorize** this image.

After each attempt you will be given correctness feedback.

*Figure 5.* The sequential prediction task as presented to Amazon Mechanical Turk workers, who 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.

sequences and averaged the correctness of the predictions across opportunity counts. We used CART for prediction because its internal structure is similar to TRESTLE's (i.e., they both construct tree structures), but learning in CART is nonincremental and optimizes for prediction of a single attribute. This similarity on all but these two dimensions makes CART a reasonable candidate for testing the importance of incrementality and multi-attribute prediction when modeling human categorization.

We applied TRESTLE to this task by having it predict each instance's success label. After each prediction, we trained it with the correct instance label in order to update its knowledge base. Similar to our approach with CFE, we performed this process with 1,000 random training sequences and averaged the correctness of the predictions across opportunity counts.

The results of average prediction accuracy for 30 sequentially presented instances can be seen in Figure 6. Human labelers appear to converge around 70% accuracy, suggesting that learning the success concept is somewhat difficult for them, 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 83% accuracy. TRESTLE performs roughly equal to the humans, with a difference in confidence intervals that is most likely due to the differences in sample size (20 humans vs. 1,000 agents).

CFE learns more rapidly, with a steeper learning curve, probably because it is nonincremental, using a previously generated grammar, and processing all training cases at each step in batch fashion. Its higher asymptotic accuracy most likely results from optimizing for prediction of a single attribute. In contrast, TRESTLE learns incrementally from the examples producing a shallower learning curve and optimizes for its ability to predict all attributes, so its overall accuracy predicting

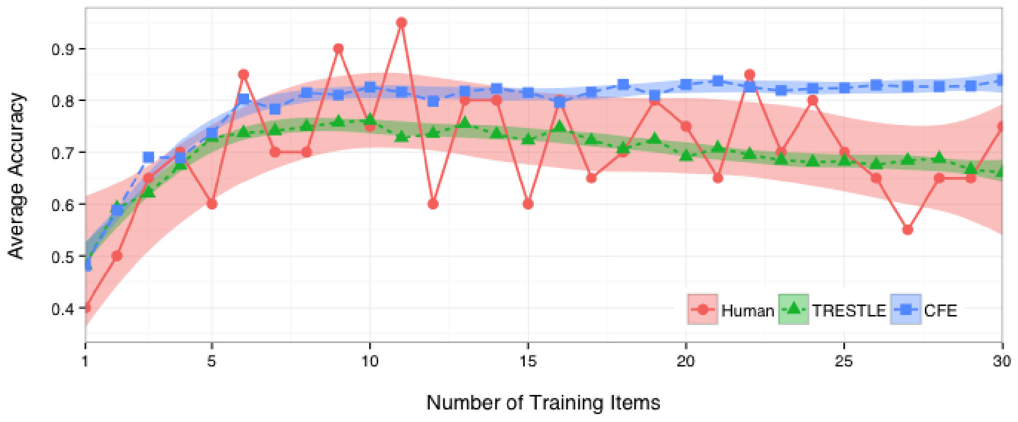


*Figure 6.* The average performance of 20 participants, 1,000 runs of TRESTLE, and 1,000 runs of CFE for predicting whether solutions to a *RumbleBlocks* level survived an earthquake (shaded regions denote the 95% Lowess confidence intervals). We see that the incremental TRESTLE algorithm performs less well than the nonincremental CFE algorithm, but that TRESTLE more closely approximates human performance.

the single attribute of success is lower. Our results suggest that, although these differences decrease predictive accuracy, they result in a model that more closely aligns with human behavior. This agrees with previous studies on human category learning, which suggest that humans often optimize for more general and flexible learning goals, and that this optimization occurs incrementally (Love et al., 2004). Overall, these findings support our claim that TRESTLE is a better model of human learning than CFE because it is incremental and it commits to being able to predict any missing attribute value. CFE may function better as a pure machine learning system and achieve higher accuracy, but it is a poorer model of human learners on the RumbleBlocks task than TRESTLE.

## 4.2 Task 2: Unsupervised Clustering

The results of the supervised learning task show that TRESTLE performs similarly to humans, but we are also interested in demonstrating its unsupervised learning capabilities and investigating whether its underlying representation of knowledge is qualitatively similar to that in humans. A strong similarity between their organizations of knowledge would strengthen our claim that it offers a better model of human learning than CFE. In particular, if its organization of concepts aligns with that of humans, it would suggest that the agreement with humans on the supervised learning task is the result of a similar organization of knowledge rather than both learners performing poorly by chance. To examine this aspect of TRESTLE, we used an unsupervised learning task in which towers were clustered without any information about their success or other category labels. For this evaluation, we chose records from three levels of *RumbleBlocks* that were part of an in-game test. These levels were attractive because they have many player solutions and because the energy ball mechanic was removed for the test setting, allowing for a wider variance of player solutions than other levels.

To establish a human baseline, we had two researchers from our laboratory independently hand cluster screenshots of all the player solutions to each level. These 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. To measure agreement between raters, we calculated the adjusted Rand index between their clusterings (Rand, 1971). This measure is a generalization of Cohen's Kappa that allows for raters to use different numbers of categories. Values range from -1 to 1, with random clusterings producing an average of zero. The average obtained across all three levels was 0.88, which is high. Given the reasonably high agreement between raters, we chose one of the human clusterings to be used in further comparisons with the machine approaches. The labeling from this rater had an average of 33 clusters across the three levels.

As a comparison, we applied the CFE process to generate feature vectors for the *RumbleBlocks* solutions. For each level, we clustered the feature vectors using the G-means algorithm (Hamerly & Elkan, 2004), which functions like a wrapper around k-means using a heuristic that tries to form clusters with a Gaussian distribution among features to choose a good value for k. This process produced a clustering of solutions for each level. We ran it ten times and we report the average and standard deviation in adjusted Rand index between the CFE and human clusterings.

To examine TRESTLE's ability to cluster *RumbleBlocks* solutions, we had it create a categorization tree for each level, with instances categorized into the tree in a random order. After all instances had been categorized once, we shuffled them and categorized them a second time, taking the concepts into which they were sorted as their labeling. We could have clustered instances using a single run, but we were interested in trying to maintain parity to the humans, who were given all the examples at once and allowed to recategorize them. To model this task we categorized the instances twice to give TRESTLE the opportunity to form an initial categorization tree from all the examples once before committing to labels.

Both the human labeling and the CFE clustering produced flat clusterings, so for comparison purposes we transformed TRESTLE's hierarchical clustering into a nonhierarchical one. As mentioned earlier, a flat clustering can be produced by splitting concepts in the tree starting at the root and returning the most general unsplit concept as a label for each instance. This process can be done with an arbitrary number of splits; for example, zero splits will return the root concept as the label for every instance, one split will return the children of the root as concept labels, and two 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 techniques. Given that the number of splits is arbitrary, we report the results for the first three splits of the categorization tree to provide a sense for how the clusterings at different levels of the hierarchy agree with human clusterings. As with the CFE evaluation, we repeated this process ten times and we report the average and standard deviation in adjusted Rand index between the human labelings and each split of the TRESTLE labelings.

Table 1 compares the two machine clusterings with the human clustering. For the Center of Mass and Wide Base levels, TRESTLE produced clusterings with the highest agreement with humans. In the Symmetry level, the CFE approach produced the most human-like clusterings, but this level had

*Table 1.* The adjusted Rand index agreement between the model clusterings and human clusterings. We report the average and standard deviation of this measure across ten clustering runs.

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

the greatest number of human clusters (40), so TRESTLE might have benefited from more splits. In general, TRESTLE seems to better match human behavior than CFE on this task.

## 5. Discussion

One interpretation of our results is that TRESTLE is a better model of human behavior while CFE is a better machine learning approach, in that it 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 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 differ, in part, because of the different motivations underlying their designs. CFE was originally created as a means of transforming instances in a structured domain into feature vectors so that other traditional learning algorithms (e.g., CART or k-means) could function in the space. It was designed primarily for use in offline settings, such as 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 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, TRESTLE maintains more psychological plausibility by avoiding exhaustive processes; for example, it does not seem very plausible that new players of *RumbleBlocks* have a preexisting *RumbleBlocks* grammar, nor that they are visually parsing towers using many grammar rules (CFE generated about 6,000 for *RumbleBlocks*). In summary, our results support the claim that TRESTLE is a better model of human behavior than CFE because it is incremental and optimizes for the prediction of any attribute.

Our results also 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 system. Further, ones that do support numeric, nominal, and relational attributes, such as FOIL (Quinlan, 1990), do not typically operate incrementally. Thus, we argue that TRESTLE is one of the few systems that supports all of these capabilities, making it a novel contribution and particularly well suited for modeling human categorization in a wide range of settings. 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 with intelligent tutors has suggested that interleaved instruction is preferable (Li et al., 2012b), but this work first induced a grammar in batch from all examples (similar to CFE), effectively pre-blocking learning of common structure. Given this pretrained structural knowledge, it makes sense that the model would learn best from interleaved problems because it only needed to identify discriminating features. In contrast, a model like TRESTLE would be well qualified to explore the issue of blocked vs. interleaved instruction, as it models the incremental acquisition of both structural information and discriminating features.

In addition to supporting multiple attribute types, our analysis demonstrates that TRESTLE supports both supervised and unsupervised learning. We argue that this makes it a more general model of learning than CFE, which employed separate machine learning algorithms (i.e., CART for prediction and k-means for clustering) for the two tasks. In the current work, we evaluated the system on separate supervised and unsupervised tasks, but TRESTLE should also support hybrid tasks that involve partial supervision. We claim that this capability makes TRESTLE ideal for modeling human learning in semi-supervised settings. In summary, our studies demonstrate the wide range of settings that TRESTLE supports and provide evidence that it is a better models human behavior than CFE in the *RumbleBlocks* domain.

## 6. Related Work

We have mentioned previously that the TRESTLE model builds on earlier accounts of concept formation in the COBWEB family (Fisher, 1987; McKusick & Thompson, 1990; Gennari et al., 1989). However, we should also discuss how it relates to these previous systems and other similar models of human categorization that attempt to accomplish similar goals. One early account of human categorization was EPAM (Feigenbaum & Simon, 1984), which modeled human memorization of nonsense syllables. It learned incrementally from sequentially presented training examples and took structural information (i.e., the organizations of letters) into account. Additionally, EPAM organized its experiences using a discrimination network, which has a tree structure similar to that used by TRESTLE. COBWEB (Fisher, 1987) extended this idea to take into account attribute-value probabilities during concept formation. Further, COBWEB/3 (McKusick & Thompson, 1990) added the ability to handle numeric information. However, these newer approaches did not support the ability to handle structural information originally supported by EPAM.

CLASSIT (Gennari et al., 1989) was another variant of COBWEB that supported both numeric information and component information. Like TRESTLE, it used both partial matching and flattening steps to handle component information during categorization. However, CLASSIT lacked support for both nominal or relational information. LABYRINTH (Thompson & Langley, 1991) was developed to support component, nominal, and relational information. Like TRESTLE, it utilized a partial matching step to rename component objects in order to maximize category utility. During categorization, the system supported components by categorizing subcomponents in a bottom-up fashion and replacing component values with nominal concept labels. LABYRINTH also generalized subcomponent labels during the categorization of higher-level components. This bottom-up categorization approach let it utilize subcomponent information to improve prediction of higher-level attributes, but it could not use higher-level component information when categorizing subcomponents. In contrast, TRESTLE's flattening mechanism takes into account the attribute-values of all components in a holistic way when making predictions. Finally, neither CLASSIT nor LABRYINTH have been compared to humans or other systems, so it is unclear how their performance would compare to that of TRESTLE.

There also exist other systems that have similar goals, but do not derive from the COBWEB family. For example, SimStudent (Li, Matsuda, Cohen, & Koedinger, 2014) shares TRESTLE's aim of modeling novice learning via a combination of statistical and structural learning. As in TRESTLE, SimStudent combines both unsupervised (perceptual representation) and supervised (skill) learning. One key difference is that TRESTLE is more unified (with a single learning mechanism) and fully incremental, whereas SimStudent has four learning mechanisms (one for representations and three for skills), with representation learning running in batch prior to skill learning. For comparison purposes, CFE is analogous to SimStudent, in that they use similar grammar learners and classification/clustering algorithms. To the extent that CFE is a good approximation of SimStudent, our results suggest that TRESTLE may model human learning better than SimStudent in that the latter, like CFE, is likely to learn faster than humans. More generally, our evidence supports the hypothesis that an incremental integration of representation and classification learning better models human behavior than separate, nonincremental, representation and classification learners. Elements of skill learning that go beyond classification (e.g., SimStudent's action-sequence learner) are targets for further research on TRESTLE.

SAGE (Kuehne, Forbus, Gentner, & Quinn, 2000; McLure, Friedman, & Forbus, 2010) is another system that shares the goal of modeling human concept formation in structured domains. It learns concepts by matching new instances to existing structures and generalizing matching concepts to include the instances. If no match is found, then it creates a new concept to cover the instance. For comparison purposes, SAGE is analogous to a form of TRESTLE that is nonhierarchical; it maintains only the children of the root and it cannot split merged concepts because it does not maintain more specific variants. This difference makes it harder for SAGE to recover from early misconceptions and biases it towards forming overly abstract concepts (Kuehne et al., 2000). Thus, in many respects TRESTLE could be viewed as an extension of SAGE that overcomes these limitations and adds support for nominal and numerical information. Another key difference is that

SAGE partially matches each instance to every concept, whereas TRESTLE only partially matches instances to its root concept.[2] This difference makes its partial matching more efficient (one match vs. many matches), but it is unclear how it affects performance. Future versions of TRESTLE should explore this tradeoff. One strength of SAGE is that it agrees with data on human behavior (Kuehne et al., 2000). It would be interesting to explore how the differences between the two models impact their agreement with observations.

In summary, the literature includes several incremental and nonincremental models of concept formation. However, each model tends to support only a subset of attribute types and rarely is their behavior compared with that of other systems or with humans. TRESTLE is novel in that it unifies facets of previous incremental approaches by supporting concept formation over nominal, numeric, structural, and relational information. It also extends systems like SAGE to support concept revision. Finally, its behavior on *RumbleBlocks* compares favorably to that found in humans.

## 7. Conclusion and Future Work

In this paper, we presented TRESTLE and demonstrated its ability to incrementally learn concepts from mixed-data environments in both a supervised and unsupervised fashion. However, our evaluation of the system is preliminary, and more work is necessary to fully assess its capabilities. In particular, we should evaluate TRESTLE's ability to perform more flexible prediction tasks (Fisher, 1987), but first, we must determine how to frame these tasks in structured domains. For example, should we evaluate a system's ability to predict specific attribute values, or should we assess its ability to predict entire missing components with multiple attribute values? This problem is complicated by structure mapping, which makes it difficult to align component predictions during evaluation.

In addition to assessing the current system's capabilities, we should integrate it with methods for planning and executing action. For example, we could extend TRESTLE to support "functional" attributes that have actions as values. This would let it decide what action to take given an instance. Unlike systems that separate concept and skill knowledge (e.g., Langley, Choi, & Rogers, 2009a), this extension would let TRESTLE use functional information to guide its formation of concepts and vice versa.

We should also explore the implications of TRESTLE for supporting designers. Prior work shows that concept formation systems can be used in this area. For example, Reich (1993) developed BRIDGER to support bridge design, but his approach did not support component or relational information. More recently, Talton et al. (2012) demonstrated how to generate novel designs using grammar patterns induced from examples provided by designers, but their work did not support mixed data types or missing attributes. TRESTLE could extend both approaches to support novel tasks, such as completing partially specified designs that are described with mixed data types. We would also like to explore how it could help game designers better understand the space of player solutions by clustering solutions with common structural patterns. This application was part of the original motivation for CFE, but our clustering results suggest that TRESTLE is better suited to the task and may even provide more information through its hierarchical organization of concepts.

2. TRESTLE compares the similarity of matched instances to its concepts during categorization, but it does not perform additional partial matching.

Finally, we should explore how we can use the system as a computational model of human learning in educational settings. In particular, we would like it to stimulate student's learning in intelligent tutoring systems and educational games. In these contexts, the model could be used to test and improve different instructional materials before they are given to students and thus determine the best ordering of activities. Additionally, TRESTLE could automatically construct domain models, in the form of concept hierarchies, through training (Harpstead, MacLellan, & Aleven, 2015). Whether they contain correct or buggy knowledge, such models could then be used in intelligent tutoring systems to provide students with improved context-sensitive feedback and instruction.

In summary, TRESTLE is a model of incremental concept formation in structured domains. It carries out supervised learning and unsupervised clustering over nominal, numeric, component, and relational information. Experiments show that it can achieve performance that is comparable to CFE, a previously developed nonincremental approach. Furthermore, although CFE achieves higher predictive accuracy, TRESTLE better matches human behavior because it takes key aspects of human categorization into account. Our studies demonstrated these capabilities and provided an initial demonstration of how it can be used to model human categorization.

## Acknowledgements

We would like to thank the designers of *RumbleBlocks* and our colleagues who conducted the formative evaluation that yielded our data. We would also like to thank Pat Langley, Victor Hwang, Kelly Rivers, Eliane Stampfer Wiese, and Douglas Fisher for their feedback and comments on earlier versions of this work. This research 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. All opinions expressed in this article are those of the authors and do not necessarily reflect the position of the sponsoring agency.

## 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., et al. (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). Louisville, KY: IEEE.

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., & Langley, P. (1990). The structure and formation of natural categories. In G. H. Bower (Ed.), *Psychology of learning and motivation: Advances in research and theory (vol. 26)*. Cambridge, MA: Academic Press.

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

Gennari, J. H., Langley, P., & Fisher, D. H. (1989). Models of incremental concept formation. *Artificial Intelligence*, *40*, 11–61.

Giraud-Carrier, C. (2000). A note on the utility of incremental learning. *AI Communications*, *13*, 215–223.

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). Cambridge, MA: MIT Press.

Harpstead, E., MacLellan, C., & Aleven, V. (2015). Discovering knowledge models in an open-ended educational game using concept formation. *Proceedings of the Sixth AIED Workshop on Intelligent Support in Exploratory and Open-ended Learning Environments*. Madrid: CEUR-WS.

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* (pp. 51–58). Memphis, TN: International Educational Data Mining Society.

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

Kellman, P. J., & Garrigan, P. (2009). Perceptual learning and human expertise. *Physics of Life Reviews*, *6*, 53–84.

Kuehne, S., Forbus, K., Gentner, D., & Quinn, B. (2000). SEQL: Category learning as progressive abstraction using structure mapping. *Proceedings of the Twenty-Second Annual Meeting of the Cognitive Science Society* (pp. 770–775). Philadelphia, PA: Cognitive Science Society.

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., Choi, D., & Rogers, S. (2009a). Acquisition of hierarchical reactive skills in a unified cognitive architecture. *Cognitive Systems Research*, *10*, 316–332.

Langley, P., Laird, J. E., & Rogers, S. (2009b). 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. (2012a). Learning to perceive two-dimensional displays using probabilistic grammars. In P. A. Flach, T. De Bie, & N. Cristianini (Eds.), *Machine learning and knowledge discovery in databases*, 773–788. Berlin: Springer.

Li, N., Cohen, W. W., & Koedinger, K. R. (2012b). Problem order implications for learning transfer. *Proceedings of the Eleventh International Conference on Intelligent Tutoring Systems* (pp. 185–194). Berlin: Springer.

Li, N., Matsuda, N., Cohen, W. W., & Koedinger, K. R. (2014). Integrating representation learning and skill learning in a human-like intelligent agent. *Artificial Intelligence*, *219*, 67–91.

Li, N., Schreiber, A. J., Cohen, W. W., & Koedinger, K. R. (2012c). Efficient complex skill acquisition through representation learning. *Advances in Cognitive Systems*, *2*, 149–166.

Love, B. C., Medin, D. L., & Gureckis, T. M. (2004). Sustain: A network model of category learning. *Psychological Review*, *111*, 309–346.

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

McLure, M., Friedman, S., & Forbus, K. (2010). Learning concepts from sketches via analogical generalization and near-misses. *Proceedings of the Thirty-Second Annual Conference of the Cognitive Science Society* (pp. 1726–1731). Austin, TX: Cognitive Science Society.

Pedregosa, F., et al. (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. (1990). Learning logical definitions from relations. *Machine Learning*, *5*, 239–266.

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.

Rau, M. A., Aleven, V., & Rummel, N. (2013). Interleaved practice in multi-dimensional learning tasks: Which dimension should we interleave? *Learning and Instruction*, *23*, 98–114.

Reich, Y. (1993). The development of Bridger: A methodological study of research on the use of machine learning in design. *Artificial Intelligence in Engineering*, *8*, 217–231.

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). Cambridge, MA: ACM.

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.

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.

Wilt, C., Thayer, J., & Ruml, W. (2010). A comparison of greedy search algorithms. *Proceedings of the Third Symposium on Combinatorial Search* (pp. 129–136). Menlo Park, CA: AAAI Press.

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 Conference on Artificial Intelligence* (pp. 864–869). Menlo Park, CA: AAAI Press.