Discovering knowledge models in an open-ended educational game using concept formation

Erik Harpstead, Christopher J MacLellan, Vincent Aleven

Human Computer Interaction Institute, Carnegie Mellon University,
5000 Forbes Avenue, Pittsburgh, PA, 15232, USA
{eharpste, cmaclell, aleven}@cs.cmu.edu

Abstract. Developing models of the knowledge and skills being exercised in a task is an important component of the design of any instructional environment. Developing these models is a labor intensive process. When working in exploratory and open-ended environments (EOLEs) the difficulty of building a knowledge model is amplified by the amount of freedom afforded to learners within the environment. In this paper we demonstrate a way of accelerating the model development process by applying a concept formation algorithm called TRESTLE. This approach takes structural representations of problem states and integrates them into a hierarchical categorization, which can be used to assign concept labels to states at different grain sizes. We show that when applied to an open-ended educational game, knowledge models developed from concept labels using this process show a better fit to student data than basic hand-authored models. This work demonstrates that it is possible to use machine learning to automatically acquire a knowledge component model from student data in open-ended tasks.

1 Introduction

When designing intelligent instructional support in educational learning environments it is important to have a model of the skills and knowledge employed during problem solving. A common approach to modeling skills in intelligent tutoring systems (ITSs) is knowledge component (KC) modeling [1]. In the KLI Framework a KC is "an acquired unit of cognitive function or structure that can be inferred from performance on a set of related tasks" [2]. A KC model is a mapping of each problem-solving step in a particular educational environment to the skills necessary to solve that step. KC models can be used in pedagogical software to drive feedback and hints, guide problem selection [3], and inform redesign of the interface [4].

While KC models are useful for a number of purposes in the development of intelligent software they take significant effort to develop. The process of creating a KC model often employs elements of empirical and theoretical task analyses [1], soliciting expert feedback and rationally constructing the skills used in a task. When working in exploratory and open-ended environments (EOLEs) this process is aggravated by the freedom learners experience in these environments. It can be assumed that as the space learners are allowed to explore grows, so too must a KC model grow to continue to provide useful support and feedback to learners. In addition to providing large spaces for exploration, EOLEs often contain more complex representations of domains making it more difficult to articulate the rules defining the applicability of a given KC.

To address the challenges of KC model creation in EOLEs we have developed a novel method for generating new KC models based only on problem states taken from the learning environment. Our approach uses a form of automated model discovery that employs a concept formation algorithm called TRESTLE [5]. This algorithm creates a hierarchical categorization tree based on training examples, which can then be used to label problem states at various grain sizes. The algorithm is designed to handle messy, mixed representations of data, making it ideal for application to EOLEs. It has previously been shown to create clusters similar to humans [5]. In this paper, we show how the conceptual patterns learned by TRESTLE can be used to discover new KC models in the open-ended educational game *RumbleBlocks* [6]. Finally, we conclude with a brief discussion of the implications of this approach and detail how we plan to expand it in future work.

2 The TRESTLE Algorithm

TRESTLE [5] is an incremental concept formation algorithm that creates a hierarchical categorization tree from a set of structured instances. In this section we briefly describe the algorithm's major structures and categorization procedure for more details see [5][1].

The TRESTLE algorithm produces a categorization tree and functions over a set of instances, each described by a set of attribute-value pairs. Instance attributes can have nominal, numeric, or component values that have their own sub-attributes and values. When integrating a new instance TRESTLE proceeds through 3 major steps:

1. Partial Matching, which renames instance attributes to align with the algorithm's current domain understanding
2. Flattening, which converts structured attributes to unstructured ones, while preserving structural information.
3. Categorization, which incorporates the instance into the knowledge base.

TRESTLE's knowledge base is an evolving category structure being built from training examples and is organized into a hierarchical tree of concepts. In building its tree, the algorithm optimizes for a heuristic called category utility, which is similar to maximizing for the expected number of correct guesses that a given concept could make about the attribute-values of a given instance. During categorization new instances are sorted into the tree. At each node in the categorization tree TRESTLE considers 4 different operations and performs whichever one would result in the highest category utility: (1) adding the instance to the best child, (2) creating a new node

[1] A reference implementation is available at: https://github.com/cmaclell/concept_formation

for the instance, (3) merging the best 2 nodes and adding the instance to the result, or (4) splitting the best node by promoting its children to be children of the current node.

After categorizing an instance into its knowledge base, TRESTLE returns a concept label for the instance. Since concepts in TRESTLE are organized in a hierarchical tree, the cluster labels returned from categorization can be generalized if more coarse clusters are desired. At the coarsest, i.e. the root of the tree, everything is considered to be the same concept, while at the most specific, i.e. the leaves of the tree, everything is considered to be unique.

To arrive at a KC label for a step, the problem state in which the step too place is categorized and label is generated based on the returned concept and a desired depth. For a given depth model the state is categorized down the TRESTLE tree. Once the state reaches the desired depth the current concept's label is returned. If the state reaches a leaf of the tree before reaching the desired depth, then the label of the deepest node is used instead. When generating KC models this allows for the creation of multiple model variants that consider the domain at different levels of granularity (see **Fig. 1**).

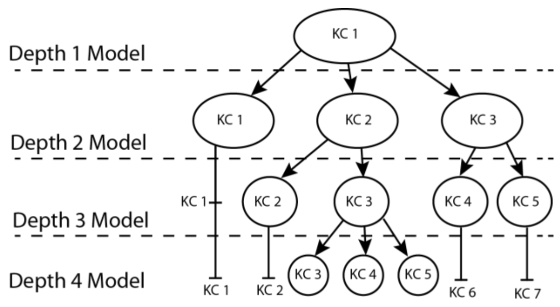

Fig. 1. A diagram of how KC labels are attributed to problem states based on their categorization in the TRESTLE tree for a given depth mode.

3 *RumbleBlocks*

To demonstrate how TRESTLE can be used to aid in the process of KC modeling we introduce *RumbleBlocks* [6], an open-ended educational game. *RumbleBlocks* is a physics game designed to teach children (ages 5-8) three basic concepts of structural stability and balance: (1) objects with wider bases are more stable, (2) objects that are symmetrical are more stable, and (3) objects with lower centers of mass are more stable.

In the game, players are tasked with building a tower out of blocks to help a stranded alien power their spaceship (see **Fig. 2**). The tower must be tall enough to reach the alien and cover a series of energy orbs that power the spaceship. Once players have finished building their tower they place the spaceship on top, which triggers an earthquake. If, after the earthquake, the ship is still on top of the tower, then the player has succeeded and advances on to the next level, otherwise they must try the level again.

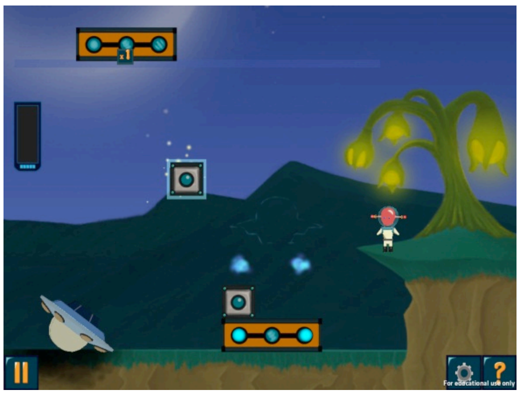

Fig. 2. A screenshot of *RumbleBlocks*

Each level in *RumbleBlocks* is designed to emphasize one of the three key concepts of stability. This emphasis is accomplished through the placement of energy orbs, the target zone for the spaceship, and the palette of available blocks. While each level is targeted at a particular principle, there is a wide range of variance in the kinds of solutions players design to in-game challenges. Our previous analysis found that there are several levels where less than 10% of students actually used the solution envisioned by the game's designers [7]. The variance in player behavior demonstrates the open-endedness of the game as well as highlights the challenge inherent in defining KC models to measure learning in the game.

4 KC Model Discovery in *RumbleBlocks*

To evaluate the application of TRESTLE to the KC modeling process we used it to discover a set of new KC models in *RumbleBlocks*. For comparison we also created a "hand built" KC model meant to capture the original design intent behind the game. This model labels each level in the game with the principle it is designed to emphasize. Since the first 5 levels of the game are primarily a mechanical tutorial for the game rather than instructional levels dealing with physics principles, we relabeled these levels with an "Intro" KC, resulting in a hand-built model with 4 KCs.

For this first demonstration of the use of TRESTLE to generate KC models we chose to focus on a broad definition of a step as solving an entire level of *Rumble-Blocks*. This is in keeping when Van Lehn *et al.*'s definition of a step as "the smallest possible correct entry that a student can make" [8] because, in its current form, *RumbleBlocks* only provides correctness feedback to players at the end of a level. In this context a step is then considered in terms of the initial level state given to the player to construct a solution in and evaluated based on their final construction. The state representation used for training TRESTLE contained the positions of each of the energy orbs, the target position of the spaceship, and the available number of each block type. The resulting categorization tree, based on the initial state data from *Rumble-*

Blocks' 47 levels, was 7 levels deep giving us 7 candidate KC models each with differing levels of granularity.

To evaluate relative appropriateness of different candidate KC models we used the tool suite provided by DataShop [9]. In particular, we used AFM [10], a specialized form of logistic regression that fits a given KC model to student log data. The resulting regression model can be used to assess the fit of a particular KC to the real student data. DataShop provides several model fit statistics to compare KC models: AIC and BIC, both standard model fit statistics that penalize for model complexity and Cross Validated Root Mean Square Error (CV-RMSE) using 3-fold cross validation with different stratification schemes (i.e. student, item and un-stratified).

The data we use in our evaluation comes from a formative evaluation of the game with 174 players in the target demographic. Players were allowed to play the game for two 20-minute sessions.

The model fit estimates for the 7 Trestle-based models and the original Principle (i.e., hand-built) model can be seen in **Table 1**. In general, more fine grained models tend to fit the data better. The TRES-Depth7 model is preferred according to AIC and both item-stratified and un-stratified RMSE. This would suggest that an appropriate model for initial states in *RumbleBlocks* is one that treats all levels as nearly unique from each other.

Table 1. Fit statistics for each KC model. Cross Validated Root Mean Square Errors (CV-RMSE) are based on 3 fold cross validation using different forms of stratification.

Model	*KCs*	*AIC*	*BIC*	*CV-RMSE (student)*	*CV-RMSE (item)*	*CV-RMSE (none)*
Principle	4	6560.73	**8544.74**	.3856	.3883	.3869
TRES-Depth1	1	6828.35	8771.45	.3924	.3948	*NA*
TRES-Depth2	5	6737.21	8734.85	.3899	.3921	.3923
TRES-Depth3	14	6661.67	8782.03	.3878	.3904	.3915
TRES-Depth4	24	6530.78	8787.50	.3845	.3853	.3855
TRES-Depth5	32	6350.50	8716.31	.3794	.3821	.3826
TRES-Depth6	39	6152.75	8614.01	**.3734**	.3761	.3739
TRES-Depth7	41	**6152.28**	8640.81	.3736	**.3754**	**.3732**

5 Discussion

We can see from the results that KC models based on depth cuts of a TRESTLE categorization tree better fit student data than a model based on the original design of the game in terms of AIC and cross-validation. According to these statistics, we find that a more specific KC model better fits student data than more general models. This would make it appear that there is little transfer going on within the game. However, this is likely due to our unit of analysis. An approach that employs a more fine grained definition of a correct step (e.g., steps defined at the transaction level) might reach a different conclusion with regards to transfer because there is likely to be some common application of knowledge components used across building towers in different levels.

The approach presented here deals with concept granularity at a holistic level. By this we mean that all KCs in a model are being considered at the same depth of the concept tree. There is some evidence that suggests human learners will employ concepts at different levels of granularity based on their expertise [11]. It is possible that the most appropriate KC model uses a combination of specific and general concepts depending on the context of the task at hand. Rather than creating KC labels as uniform cuts of a concept hierarchy, where concepts all exist at the same depth, we could instead start all problem states at their coarsest label and iteratively split concept nodes into more specific labels. After each split the resulting KC model could be tested for fit using student data until an optimal model is found. This is similar to the Learning Factors Analysis search algorithm [12] but it would not require human developed models as seeds. Exploring this process is something we look forward to in future work.

Our current analysis defined steps to be the complete solution to each level. This follows with Van Lehn *et al.*'s definition of a step in KC analysis as the smallest amount of action that a student can perform correctly [8]. This definition still assumes that all possible solutions to a level exercise the same skill, which may not be the case in practice. One way of going beyond this assumption in analyzing *RumbleBlocks* is to create a TRESTLE model based on the solutions players make to each in-game level rather than the initial conditions of the level. Such an approach would allow for analysis according to different kinds of solutions rather than the constraints under which problem solving took place. One issue with taking into account the content of students' solutions is how to handle the assignment of KC labels when there are multiple valid solutions to a level, as is the case with *RumbleBlocks* [7]. In the case of correct solutions it is simple to state that each unique correct solution embodies the use of a different KC. When looking at incorrect solutions, however, the question of attribution becomes more difficult as it is hard to know which of the possible correct approaches the student failed to execute correctly. A standard modeling approach would assign an incorrect step with the labels of all possible correct solutions; using a variant to AFM's statistical formula to allow for the disjunction of KCs [10]. A TRESTLE based approach could go beyond this by categorizing incorrect solutions into a knowledge base trained on correct solutions and assigning a KC label based on which correct solution the error most closely resembles. This is similar to the approach taken by Rivers and Koedinger to create next step feedback in programing tasks [3] but has the potential to be domain general. Exploring this approach to KC modeling with TRESTLE remains a topic of our future work.

Ideally, we would like to go beyond the final state definition of a step to a transaction-level model. Having a full transaction-level model would allow for the inclusion of targeted feedback to players while they are playing rather than providing feedback only at the end of building. Additionally, more detailed understanding of player problem solving could better inform adaptive sequencing. The challenge in taking this approach in *RumbleBlocks* is that that evaluation of player performance is currently only performed at the end of a level. This creates similar correctness attribution challenges in deciding whether a particular build step is a good or bad example of a given concept. Again we could turn to TRESTLE to aid in this analysis by having it perform categorization on whole solution paths rather than final solutions. There are several open questions with this analysis in terms of how best to represent a solution path for categorization but we hope to resolve these issues in future work.

6 Conclusion

This paper presents a preliminary use of TRESTLE as a way to discover new KC models in an open-ended game. The models automatically discovered by TRESTLE better fit student data than one hand-built to capture the design intent of the game. This demonstrates the promise of concept formation based approaches to KC model creation. In future work we plan to further explore the implications of TRESTLE-based KC models including discovering transaction-level models and exploring models that capture mixed grain sizes. We hope other researchers can find utility in these methods and apply them to their own exploratory and open-ended environments.

7 Acknowledgement

We would like to thank the developers of *RumbleBlocks* and our colleagues who performed the evaluation that provided our data. This work was supported in part by the DARPA ENGAGE research program under ONR Contract Number N00014-12-C-0284 and by a Graduate Training Grant awarded to Carnegie Mellon University by the Department of Education # R305B090023.

8 References

1. Aleven, V., Koedinger, K.R.: Knowledge Component Approaches to Learner Modeling. In: Sottilare, R.A., Graesser, A., Hu, X., and Holden, H. (eds.) Design Recommendations for Intelligent Tutoring Systems: Volume 1 - Learner Modeling. pp. 165–182. U.S. Army Resarch Laboratory (2013).
2. Koedinger, K.R., Corbett, A.T., Perfetti, C.: The Knowledge-Learning-Instruction Framework: Bridging the Science-Practice Chasm to Enhance Robust Student Learning. Cogn. Sci. 36, 757–98 (2012).
3. Rivers, K., Koedinger, K.R.: Automating Hint Generation with Solution Space Path Construction. Proc. ITS 2014. pp. 329–339 (2014).
4. Koedinger, K.R., Stamper, J.C., Mclaughlin, E.A., Nixon, T.: Using Data-Driven Discovery of Better Student Models to Improve Student Learning. Proc. AIED 2013. pp. 421–430 (2013).
5. Maclellan, C.J., Harpstead, E., Aleven, V., Koedinger, K.R.: TRESTLE: Incremental Learning in Structured Domains using Partial Matching and Categorization. Proceedings of the 3rd Annual Conference on Advances in Cognitive Systems - ACS 2015 (2015).
6. 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., Chase, C.,

Harpstead, E., Lomas, D.: RumbleBlocks: Teaching science concepts to young children through a unity game. Proc. CGAMES 2012. pp. 162–166. IEEE (2012).

7. Harpstead, E., Maclellan, C.J., Koedinger, K.R., Aleven, V., Dow, S.P., Myers, B.A.: Investigating the Solution Space of an Open-Ended Educational Game Using Conceptual Feature Extraction. Proc. EDM 2013. pp. 51–58 (2013).
8. Vanlehn, K., Koedinger, K.R., Skogsholm, A., Nwaigwe, A., Hausmann, R.G.M., Weinstein, A., Billings, B.: What's in a Step? Toward General, Abstract Representations of Tutoring System Log Data. Proceedings of the 11th International Conference on User Modeling - UM 2007. pp. 455–459. Springer Berlin Heidelberg (2007).
9. Koedinger, K.R., Baker, R.S.J. d, Cunningham, K., Skogsholm, A., Leber, B., Stamper, J.: A Data Repository for the EDM community: The PSLC DataShop. In: Romero, C., Ventura, S., Pechenizkiy, M., and Baker, R.S.J. d. (eds.) Handbook of Educational Data Mining. pp. 43–55 (2010).
10. Cen, H., Koedinger, K., Junker, B.: Comparing Two IRT models for conjunctive skills. Proc. ITS 2008. pp. 796–798 (2008).
11. Fisher, D.: A Computational Account of Basic Level and Typicality Effects. Proc. 1988 Nat. Conf. Artificial Intelligence (AAAI'88). pp. 233–238 (1988).
12. Cen, H., Koedinger, K., Junker, B.: Learning Factors Analysis – A General Method for Cognitive Model Evaluation and Improvement. Proceedings of the 8th International Confernence on Intelligent Tutoring Systems - ITS 2006. pp. 164–175 (2006).