Scaling cognitive modeling to massive open environments

Yanbo Xu YANBOX@BERKELEY.EDU
University of Carlifonia Berkeley

Matthew J. Johnson MATTJJ@CSAIL.MIT.EDU
Harvard University

Zachary A. Pardos ZP@BERKELEY.EDU
University of Carlifonia Berkeley

Abstract

The challenges in scaling cognitive modeling to massive open online environments often involve algorithmic tractability and knowledge representation for the complex subjects presented online. To solve the computational tractability issue, we present an optimized c++ implementation of student models with parallelized Baum-welch fitting method. To solve the knowledge representation issue, we describe a hierarchical cognitive model based on courseware structure, and propose an extensive knowledge tracing model to incorporate students' activities on the platform. We evaluate the models on three online courses with cross validation, and report the results in terms of Root Mean Square Error (RMSE).

1. Introduction

The practice of inferring student knowledge and learning in massive open online environments faces two major issues of scale; algorithmic tractability and how to represent knowledge in the multitude of complex subject matter found online, particularly in college level offerings. In this paper we introduce an open-source solution to the computational tractability issue with an optimized c++ implementation of our model that includes parallelized Baum-welch for parameter learning. We approach the knowledge representation issue by deriving six different representations from courseware structure each at a different level of conceptual granularity. We use cross-validation to empirically evaluate which representation of knowledge best explains students' longitudinal question level performance in the course. Lastly, we apply an extension of the cognitive model in order to incorporate the breadth of activities conducted by students on the platform and measure the impact of those activates on knowledge gained in the dimensions of the six modeled knowledge representations. Our results from applying this technique to three Massive Open Online Courses suggest that attribution of learning to various courseware elements improves predictive performance. Implications of this work include delivering inferred resource pedagogical efficacy information to instructors and providing automatically generated resource pointers to learners.

Workshop on Machine Learning in Education at the 32^{nd} International Conference on Machine Learning, Lille, France, 2015. Copyright 2015 by the author(s).

2. Data set

We include three datasets consists of tracking log data respectively from Maggie Sokolik's BerkleyX course, "Principles of English Writing, Part I" (**ColWri**) offered on edX in the Fall of 2013; Armando Fox, David Patterson, and Sam Joseph's BerkleyX course, "Engineering Software as a Service" (**CS**) offered on edX in August of 2013; Ani Adhikari and Philip B. Stark's BerkleyX course, "Introduction to Statistics: Inference" (**Stats**) offered on edX in the Spring of 2013.

The **ColWri** course is organized into five chapters, consisted of 43,766 enrolled participants producing 7.6 million events; the **CS** course is organized into nine chapters, consisted of 8,978 enrolled participants producing 3.6 million events; the **Stats** course is organized into five chapters, consisted of 46,067 enrolled participants producing 18.6 million events. Figure 1 shows the number of unique participants that logged at least a single action in each day of three the course. The attrition was customarily steep but less so than the 17 initial MITx and HarvarX courses (Ho et al., 2014) which experience a 90% drop-off in the first week. For example, of the 43 thousand enrolled in the **ColWri** course, 4,600 (10%) received certification. There were 80 assessment questions spread over 28 different problems. In the edX platform, a problem typically consists of a figure or passage of text and subsequent related questions.

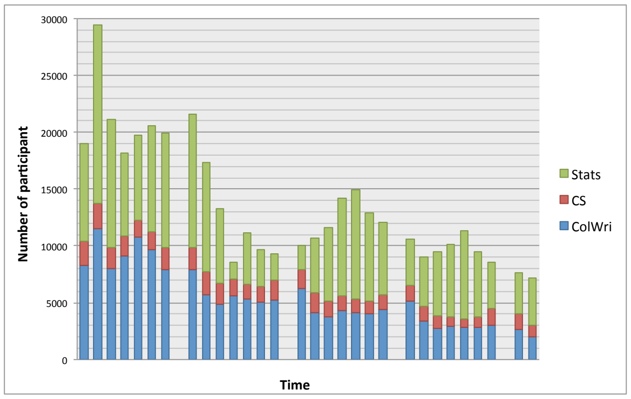


Figure 1. Number of participants over time

Thirty five verb described each logged action in our event table that was compiled to include the actor,verb,object format found in xAPI/TinCan (Initiative, 2013). The 35 verbs and their frequencies are organized into groups of six verb types in Figure 2.

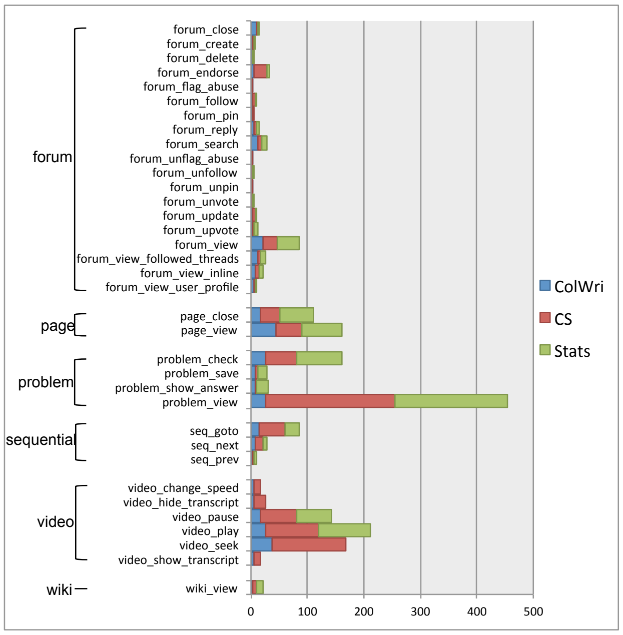


Figure 2. Frequency per action types per participant

3. Knowledge representation

An intuitive knowledge representation is to assume a unified knowledge component (KC) that generalizes the entire course. Another simple representation can assume individual KCs underlining each individual conceptualized knowledge units. Sophisticated knowledge representation usually involve efforts from human experts in the domain to develop. With the multitude of complex subject matter found in massive open online environments, especially in college level offerings, knowledge representation becomes even harder. To approach this issue, we generate hierarchical KC models from the courseware structure each at a different level of conceptual granularity.

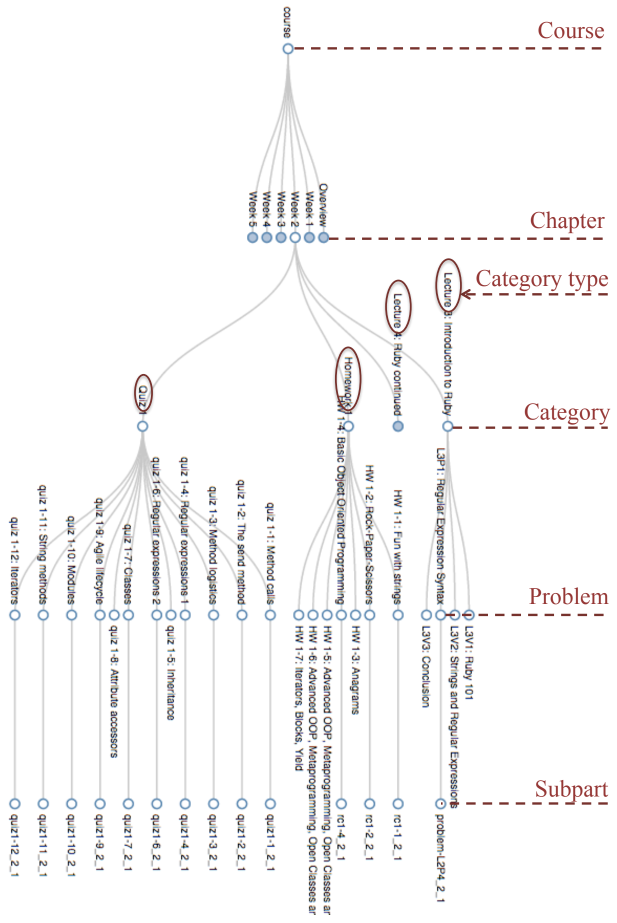

course
Course
Week 5
Week 4
Week 3
Week 2
Week 1
Overview
Chapter
Category type
Lecture 3: Introduction to Ruby
Lecture 4: Ruby continued
Homework 1
Quiz 1
Category
quiz 1-12: Iterators
quiz 1-11: String methods
quiz 1-10: Modules
quiz 1-9: Agile lifecycle
quiz 1-8: Attribute accessors
quiz 1-7: Classes
quiz 1-6: Regular expressions 2
quiz 1-5: Inheritance
quiz 1-4: Regular expressions 1
quiz 1-3: Method logistics
quiz 1-2: The send method
quiz 1-1: Method calls
HW 1-7: Iterators, Blocks, Yield
HW 1-6: Advanced OOP, Metaprogramming, Open Classes ar
HW 1-5: Advanced OOP, Metaprogramming, Open Classes ar
HW 1-4: Basic Object Oriented Programming
HW 1-3: Anagrams
HW 1-2: Rock-Paper-Scissors
HW 1-1: Fun with strings
L3V3: Conclusion
L3P1: Regular Expression Syntax
L3V2: Strings and Regular Expressions
L3V1: Ruby 101
Problem
Subpart
quiz1-12_2_1
quiz1-11_2_1
quiz1-10_2_1
quiz1-9_2_1
quiz1-7_2_1
quiz1-6_2_1
quiz1-4_2_1
quiz1-3_2_1
quiz1-2_2_1
quiz1-1_2_1
rc1-4_2_1
rc1-2_2_1
rc1-1_2_1
problem-L2P4_2_1

Figure 3. Generative structural cognitive models

The strategy is to start with a unified KC for the entire course, and then refine it by traversing down the courseware structure. Figure 3 shows a tree-like courseware structure of the “Principles of English Writing, Part I” course, and annotates each level of concepts as different KC levels. Thus the KC models, ranging from **Course** to **Subpart**, categorize the minimal concepts defined at the leaves with different level of granularity. A **Course** level model gives the unified KC model. A **Chapter** level model assume the subparts within the same chapter share the same level of KC. In edX system, the chapters of a course are usually partitioned by week. So the Chapter level model include KCs such as *Week1*, *Week2* and so on. The **Category type** level summarizes the types of **Category**, and gives KCs such as *Lecture*, *Homework*, *Quiz* etc; while the **Category** level shows which specific lecture or homework or others the subparts belong to, and gives KCs such as *Lecture1, 2,...*, *Homework1, 2,...*, etc. A **Problem** level model specifies which problem in a lecture or homework or others the subparts belong to, e.g. *Lecture1Problem1, Lecture1Problem 2, ..., Homework1Problem 1, Homework2Problem1,...*, etc. At last, the **Subpart** level enumerates the minimal level of concepts defined in a Problem, and gives KCs such as *Lecture1Problem1_1, Lecture1Problem1_2,..., Homework1Problem 1_1, Homework2Problem1_1,...*, etc. We list the KC models at different level of conceptual granularity as follows, where # denotes the nonnegative numbering:

- **Course**: *Course.*
- **Chapter**: *Overview, Week#.*
- **Category type**: *Lecture, Homework, Quiz, Exam.*
- **Category**: *Lecture#, Homework#, Quiz#, Exam#.*
- **Problem**: *L#P#, HW#-#, Quiz#-#, Exam#-#*
- **Subpart**: *L#P#_#, HW#-#_#, Quiz#-#_#, Exam#-#_#*

4. Computational model

We use Bayesian Knowledge Tracing (BKT) to model students' learning and discover students' hidden knowledge based on their responses to the subparts recorded in our edX data. With no pre- and post-testing, BKT can help us to identify when a student have already mastered the knowledge component. And it also can help to predict a student's future performance given the student's history of practices. So, we first give a short review of the standard BKT.

4.1. A review of the standard BKT

Bayesian Knowledge Tracing (BKT) (Corbett & Anderson, 1994) has been widely applied in intelligent tutor systems and educational data mining. Figure 4 shows the graph of standard BKT, where the unobserved nodes represent students' hidden knowledge and the observed nodes represent students' performance.

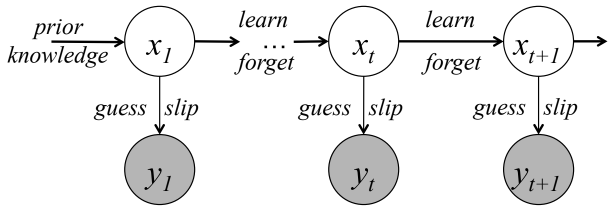


Figure 4. Standard Bayesian Knowledge Tracing (BKT)

More specifically, the binary hidden states denote the student (*knowing* or *not knowing*) and the binary observed states denote the student (*correct* or *incorrect*) performance. BKT models the student's learning rate as a transition probability from *not knowing* at the previous step to *knowing* at the current step, the student's guessing rate as the emission probability from *not knowing* to *correct* performance, and the student's slipping rate as the emission probability from *knowing* to *incorrect* performance at the step.

Parameters are formally defined in the following formulas:

$$\textit{prior knowledge} = \Pr[x_1 = 1] \tag{1}$$
$$\textit{learn} = \Pr[x_{t+1} = 1 | x_t = 0] \tag{2}$$
$$\textit{guess} = \Pr[y_t = 1 | x_t = 0] \tag{3}$$
$$\textit{slip} = \Pr[y_t = 0 | x_t = 1] \tag{4}$$

Given the above parameters, students' performance at time t can be predicted by

$$\Pr[y_t = 1] = \Pr[x_t = 1] \cdot \neg \textit{slip} + Pr[x_t = 0] \cdot \textit{guess}. \tag{5}$$

We assume an online inference, which updates the student's knowledge right after observing the student's performance. That is, when observing the student does *correct* at time t, we update the knowledge at time t as:

$$\Pr[x_t = 1 | y_t = 1] = \frac{\Pr[x_t = 1] \cdot \neg \textit{slip}}{\Pr[x_t = 1] \cdot \neg \textit{slip} + Pr[x_t = 0] \cdot \textit{guess}}; \tag{6}$$

and when observing *incorrect* at time t, we update:

$$\Pr[x_t = 1 | y_t = 0] = \frac{\Pr[x_t = 1] \cdot \textit{slip}}{\Pr[x_t = 1] \cdot \textit{slip} + Pr[x_t = 0] \cdot \neg \textit{guess}}. \tag{7}$$

At last, the knowledge at time $t+1$ is inferred as:

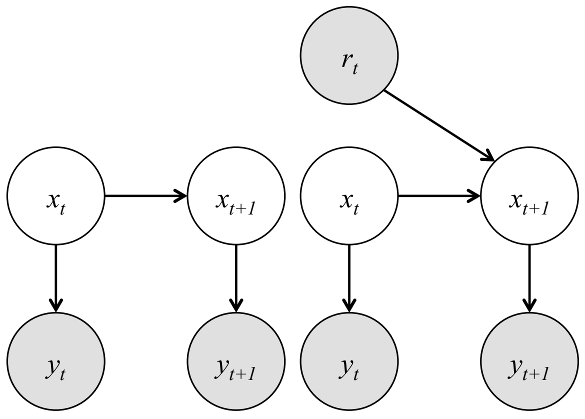
$$\Pr[x_{t+1} = 1] = \Pr[x_t = 1 | y_t] + \Pr[x_t = 0 | y_t] \cdot \textit{learn}. \tag{8}$$

Standard BKT can be simply represented as a Hidden Markov Model (HMM), as shown in Figure 5(a). We add a discrete state r_t, which takes multiple values that enumerate the "resources" defined from the data, as shown in Figure 5(b). An early pilot of this model was discussed in (Pardos et al., 2013). We add an edge between the resource state and the next knowledge state so that the transition probabilities between two knowledge states are now depending on the last resource value. That is, the new model reserves most of the parameters from the standard BKT except *learn*, which is now fitted individually for each individual resource:

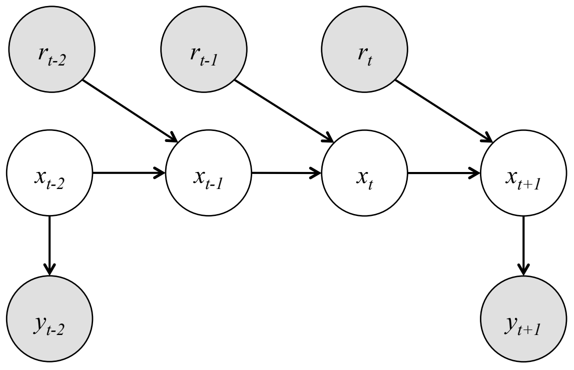
4.2. Add resource effect into BKT

Standard BKT can be simply represented as a Hidden Markov Model (HMM), as shown in Figure 5(a). We add a discrete state r_t, which takes multiple values that enumerate the "resources" defined from the data, as shown in Figure 5(b). We add an edge between the resource state and the next knowledge state so that the transition probabilities between two knowledge states are now depending on the last resource value. That is, the new model reserves most of the parameters from the standard BKT except *learn*, which is now fitted individually for each individual resource:

$$learn_r = \Pr\left[x_{t+1} = 1 | x_t = 0, r_t = r\right] \tag{9}$$



(a) HMM (Standard BKT) (b) 1 prior resource-HMM



(c) 3 prior resource-HMM

Figure 5. Graphical representation of BKT models. *x*'s: binary hidden knowledge states; *y*'s: binary observed performance states; *r*'s: multi-valued discrete observed resource states

Prediction and online inference are still the same as before, but the knowledge at time $t+1$ is now inferred from which resource was observed at time t:

$$\begin{aligned}&\Pr\left[x_{t+1} = 1 | r_t = r\right] \\ &\quad = \Pr\left[x_t = 1 | y_t\right] + \Pr\left[x_t = 0 | y_t\right] \cdot learn_r. \end{aligned} \tag{10}$$

So we create a new BKT model, discovering students' hidden knowledge not only from their observed performance but also what resources the students were using previously. In contrast to BKT models, related work (Lan et al., 2014) uses an approximate Kalman filter approach to jointly trace students' hidden knowledge evolution and estimate the quality of the corresponding learning resources.

We could include all the resources that students have visited after their last responses and before the current responses. Alternatively, we only include the n "closest" resources prior to the current response, which are considered to be the most likely relevant resources to the current problem. For example, Figure 5(c) shows a $n = 3$ prior resource model. Each intermediate state between two observed responses get updated each time when a resource is observed. The knowledge at time $t+1$ can be recursively updated as:

$$\Pr\left[x_{t+1} = 1 | r_t = r_3, r_{t-1} = r_2, r_{t-2} = r_1\right] \tag{11}$$
$$= \Pr\left[x_t = 1 | r_{t-1} = r_2\right] + \Pr\left[x_t = 0 | r_{t-1} = r_2\right] \cdot learn_{r_3},$$
$$\Pr\left[x_t = 1 | r_{t-1} = r_2, r_{t-2} = r_1\right] \tag{12}$$
$$= \Pr\left[x_{t-1} = 1 | r_{t-2} = r_1\right]$$
$$+ \Pr\left[x_{t-1} = 0 | r_{t-2} = r_1\right] \cdot learn_{r_2},$$
$$\Pr\left[x_{t-1} = 1 | r_{t-2} = r_1\right] \tag{13}$$
$$= \Pr\left[x_{t-2} = 1 | y_{t-2}\right] + \Pr\left[x_{t-2} = 0 | y_{t-2}\right] \cdot learn_{r_1}.$$

Next, we introduce how to scale these BKT models to MOOC data. We first describe the current HMM fitting algorithms and their scalabilities, and then present our optimized C++ implementation with parallelization, which we call *xBKT*[1]. Related work (González-Brenes et al., 2014) provides an alternative optimized Java implementation of BKT models, but with no parallelizations.

4.3. Scale BKT models to MOOC (*xBKT*)

4.3.1. HMM Fitting Algorithms and Scalability

Almost all standard HMM fitting algorithms alternate between two steps: (1) dynamic programming over data sequences and (2) model updates. For example, in the context of the ubiquitous Expectation-Maximization (EM) algorithm for finding local optima of the data log likelihood, (1) is the E step and (2) is the M step (Rabiner, 1989; Bishop, 2006). These same two steps are present in methods motivated from both Frequentist and Bayesian frameworks, including Viterbi EM, (stochastic) gradient descent on log likelihood, Gibbs sampling (Johnson & Willsky, 2013), variational mean field (Beal, 2003), and stochastic variational methods (Hoffman et al., 2013). In all of these

[1]https://github.com/CAHLR/xBKT

cases, alternating fitting procedures arise because of the HMM model structure: given some estimate of the model, the Markov structure on the state sequence enables efficient dynamic programming inference over sequential data. The same alternating algorithm design, and hence the same scaling ideas, apply also to hierarchical models built using HMMs.

Here, we focus on the standard EM algorithm for model fitting and use it to highlight basic scaling issues for all such models. In this section we use only the time-homogeneous HMM to simplify notation, but the same ideas readily generalize to our models in which there may be multiple observations per time step and transitions depend on resources.

A basic HMM with N hidden states over a data sequence $y_{1:T}$ and a state sequence $x_{1:T}$ is parameterized by the triple (A, B, π) where

$$A_{ij} := \Pr\left[x_{t+1} = j | x_t = i\right] \tag{14}$$

$$B_{ij} := \Pr\left[y_t = j | x_t = i\right] \tag{15}$$

$$\pi_i := \Pr\left[x_1 = i\right] \tag{16}$$

for all $t = 1, 2, \ldots, T$ and $i, j = 1, 2, \ldots, N$. In the E step, we compute the expected statistics for the distributions parameterized by (A, B, π) conditioning on the data $y_{1:T}$; for example, the expected statistics for the transition matrix A are

$$\begin{aligned} &\sum_{t=1}^{T-1} [\mathbf{1}[x_t = i, x_{t+1} = j]] \qquad (17)\\ &= \sum_t \Pr\left[x_t = i, x_{t+1} = j | y_{1:T}\right] \\ &\propto_{i,j} \sum_t \underbrace{\Pr[y_{1:t}, x_t = i]}_{=\alpha_t(i)} \underbrace{\Pr[x_{t+1} = j | x_t = i]}_{=A_{ij}} \\ &\quad \underbrace{\Pr[y_{t+1:T} | x_{t+1} = j]}_{\propto \beta_{t+1}(j) B_{i y_{t+1}}} \end{aligned}$$

where the expectation is over $p(x_{1:T}|y_{1:T})$ and where α and β are the forward and backward messages, respectively, which can be computed efficiently in $\mathcal{O}(TN^2)$ time and $\mathcal{O}(TN)$ space each via recursion, e.g.

$$\alpha_{t+1}(i) = \sum_j \alpha_t(j) A_{ij} B_{i y_{t+1}}. \tag{18}$$

The computation for the other statistics is analogous, and the M step simply renormalizes these statistics.

Therefore the algorithm consists of iterating a forward and backward pass over each data sequence to collect statistics with aggregation and normalization of those statistics. Since the passes over each sequence are independent of one another, and since both the model parameters and the collected statistics are relatively small, it is natural to run the E step in parallel for many sequences and aggregate the results via reduction. This parallelization strategy works well for both multicore (shared memory) machines as well as cluster and cloud computing settings.

Given a model fit, we can evaluate the fit's prediction accuracy by calculating one-step-ahead predictions on test data using the forward message recursion:

$$\Pr[y_{t+1}|y_{1:t}] = \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{\alpha_t(i)}{\sum_{k=1}^{N} \alpha_t(k)} A_{ij} B_{j y_{t+1}}. \tag{19}$$

4.3.2. C++ Implementation

To enable scaling to large data while keeping model fits and human loops tight, we need a parallel implementation that can scale across cores and nodes as well as high-performance serial code to be run on each core.

For serial performance, we sued C++ and the Eigen template library (Guennebaud et al., 2010) for fast matrix operations. Eigen provides lazy expression templates and generates highly compiler-optimizable code; it also provides explicit vectorization (SSE2/3/4 and SIMD instructions) for extremely high performance. Critically, our implementation also optimizes memory and cache access, particularly by using the alpha-gamma recursion (a variant on the alpha-beta recursion) that eliminates extra data accesses on the second pass of the E step. We also use normalized messages instead of log messages, since divisions use fewer cycles and processor dispatch slots (Intel Corporation, 2013).

For multicore scaling, we use OpenMP (see Appendix) for lightweight threads that share a virtual address space and thus avoid costly interprocess communication. Since each of our data sequences is short, to avoid false sharing in the cache and other parallel memory allocation issues, we allocate packet-aligned temporaries on each thread's stack and thus avoid any dynamic memory allocation during the message passing step. Finally, with OpenMP 4.0 in the unreleased GCC 4.9, user-defined reductions provide efficient aggregation of statistics across threads, even as models grow and require more data to be communicated.

To handle truly massive datasets, we are experimenting with both cluster and EC2 computing using IPython parallel (Bernard, 2013) and Julia (Bezanson et al., 2012). In particular, Julia is designed for easy and flexible scaling (on both private clusters and EC2) and provides an easy interface to our low-level code. Julia itself is very efficient due to its JIT-compiled design using the LLVM compiler infrastructure.

4.3.3. Performance improvements

We test the performance of this multicore parallelized and optimized C++ implementation on our x86_64 linux server with memory of 819 gigabytes, and 4 Intel ® Xeon ® Processors each has 12 cores. We compare our C++ implementation with Murphy's *BNT* Matlab EM algorithm implementation (Murphy, 2001) that is the most widely used toolkit for BKT models in the community of educational data mining and intelligent tutor system (Chang et al., 2006; Xu & Mostow, 2011). We train the standard BKT model at KC level of *Subpart* by the full dataset of "Introduction of Statistics: Inference", which includes 1,675,985 submitted subparts from 46,067 enrolled participants.

With only one initialization and maximum iteration number of 100 for the EM algorithm, the un-parallelized single core *xBKT* takes 6.4463 seconds to converge at tolerance value of $1e-3$. The parallelized multicore *xBKT* only needs 3.1007 seconds. The *BNT* toolkit takes up to 20 hours (72,690.5819 seconds) to converge with the same EM criteria. Thus, our multicore and optimized C++ implementation is about 10,000X faster than the *BNT* Matlab implementation.

In addition, Figure 6 plots the elapsed time vs. the maximum number of threads that can be used for the parallel region in OpenMP. For our 4 processors server with 12 cores per processor, *xBKT* reaches its best performance (elapsed time of 3.1007 seconds) when the maximum number of threads is set to be 12. The elapsed time for single thread in the multicore setting is 6.5398 seconds, slightly longer than the single core setting (6.4463 seconds). It increases to 3.3492 seconds for maximum of 24 threads. The elapsed time changes nonlinearly due to the system overhead.

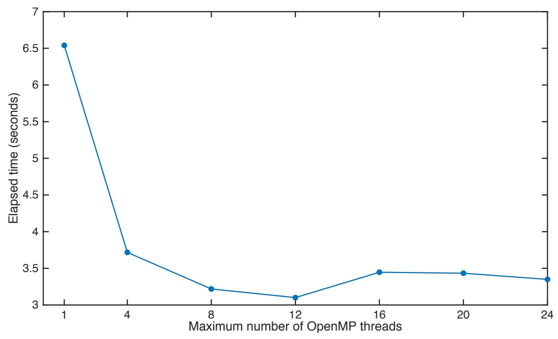


Figure 6. Multi-core *xBKT* performance measurements on **Stats** course (4 processors each has 12 cores)

5. Experiment

We use *xBKT* to fit different BKT models (with and without resources at different KC level) on three of our datasets, **ColWri** for the "Principles of Written English, Part I", **CS** for "Engineering Software as a Service" and **Stats** for "Introduction to Statistics: Inference". We predict the responses of the subparts submitted by unseen students with 5-fold cross validation, and then compare the results with respect to Root Mean Square Error (RMSE).

5.1. Methodology

5.1.1. Data pre-processing

In our data, students' responses were recorded every time when the action of "problem check" happens. Students can check in their answers to a problem multiple times. Each submission can be an empty answer, a complete answer of the entire problem, or a partial answer to one or more subparts in the problem. We ignore all the empty answers that were evaluated as *incorrect*. We also ignore all the *incorrect* responses that have the same answers submitted before. We get rid of all the consequent checks from a student after the student's first *correct* response, so that a *correct* can only happen at the end of a student's sequential attempts on a subpart. At last, we filter out the resources that was recorded less than 5 seconds ahead a student's submission in order to exclude those unconvincing resource data points.

After pre-processing the three datasets, we obtain 7,816,175 records (including 486,403 submitted subparts) for **ColWri** course, 3,562,083 records (including 188,429 submitted subparts) for **CS** course, and 18,693,393 records (including 1,675,985 submitted subparts) for **Stats** course.

5.1.2. Cross validation

For each of the datasets, we apply a 5-fold cross validation by randomly partitioning the students into 5 equal size folds. Each time we train the models on approximately 80% of the students, and then validate on the other 20% unseen students.

5.1.3. Baseline

For each KC model we defined in Section 3, the baseline is calculating the percentage of *correct* responses for each specific KC in the training set, and use it to predict the probability of *correctly* answering a subpart requiring the corresponding KC in the test set.

5.1.4. Model configuration

We fit *learn* rates per KC specified with or without resource specified, and *guess*, *slip* rates per subpart. Based on the different KC levels discussed in Section 3 and the different types of recourses that are taken into account, we need to characterize 4 properties for our BKT models.

First, we need to specify the KC models that are defined from the hierarchical courseware structure. There are 6 different KC levels, starting from *Course* level and down to *Subpart* level. Second, we need to decide at most how many prior resources should be traced back given a submitted response. Here we consider 5, 3, and 1 single prior resource(s). Note that 0 prior resource represents the standard BKT. Third, we need to specify the resource granularity, such as shall we include any of the resources that is a video, or only the videos that are for Lecture 2? In our data, there are three columns that can be used to represent the resource granularity: taking *video* as an example, *object* describes which video for which lecture should be considered; *action* describes from a another perspective, such as if this video resource is playing or paused; and *action type* describes the types of the resource, such as if this is a video or a problem. Last, we need to define the scope of the resources, such as a resource should be enclosed if it's in the same problem or the same chapter as the submitted subpart belongs to. We list the 4 properties and their specific values as below:

- KC model
  - *Subpart, Problem, Category, Category type, Chapter*, and *Course*
- Number of prior resources:
  - *5*, *3*, and *1* (*0* for the standard BKT)
- Resource granularity
  - *object* (*thread#, L#P#, L#V#*, etc.)
  - *action* (*forum_endorse, forum_view, problem_view, video _play, video_pause*, etc.)
  - *action type* (*forum*, *page*, *problem*, *sequential*, *video*, and *wiki*)
- Resource scope
  - *problem, chapter, category*, and *course*

Thus one BKT model configuration can be "*Subpart* KC model, *5* prior resources, *action type* as resource, with resources in the same *problem*". After enumerating all the combinations, we result at $6 \times 3 \times 3 \times 4 = 216$ models. Besides, we have 6 standard BKT models (equally as "*0* prior resources") and 6 baseline models based on the 6 different KC levels.

5.2. Result

5.2.1. Model fit with cross validation

For each dataset, Table 1 aggregates the RMSE per number of prior resources, across all the models with different KC levels, Resource granularity and Resource scope. Adding resource effect into BKT models significantly improves the prediction comparing to the standard models, but the number of prior resources doesn't differentiate the performance very much. Table 2 aggregates the RMSE for each Resource granularity, across all the models with different KC levels, Number of prior resources and Resource scope. The *action* resource model consistently outperforms the *object* and *action type* models, as well as the standard BKT, for all of the three datasets.

Table 1. Aggregate RMSE per Number of prior resources

		Course		
		ColWri	**CS**	**Stats**
	5	0.3149	0.2156	0.2395
Number of	3	0.3149	0.2155	0.2395
prior resources:	1	**0.3148**	**0.2153**	**0.2394**
	0 (BKT)	0.3159	0.2156	0.2397

Table 2. Aggregate RMSE per Resource granularity

		Course		
		ColWri	**CS**	**Stats**
	Object	0.3149	0.2158	0.2397
Resource	Action	**0.3145**	**0.2151**	**0.2393**
granularity:	Action type	0.3152	0.2155	0.2395
	Null (BKT)	0.3159	0.2156	0.2397

Now we set the Number of prior resource to be 1 and the Resource granularity to be *action*, and show the RMSE score per Resource scope per KC level in Table 3. The italicized numbers represent the best RMSE for each KC model, while the bold numbers represent the best RMSE for each Resource scope. First of all, all the BKT models beat our baseline for all the three datasets. Secondly, all the resource based BKT models perform no worse than the no resource (standard) BKT models. Moreover, from all of the three datasets, the best KC model is at *Problem* level (except *Subpart* for baseline), and the best Resource scope is to include resources from the same *Problem*.

5.2.2. Learning parameter

In addition to model fit, it is also interesting to see how the fitted parameters help to understand the student activities on the platform and measure the impact of those activities on knowledge gained. We set KC at *chapter* level, use *action* as the resource, include all the resources in the *course*, and trace back 5 prior resources. We fit this model to all the three full datasets and get a list of learning rates per *action* per *chapter*. We select the top 10 resources with the highest learning rates for each week, and categorize them by *video*, *webpage*, *Discussion*, *Homework/Quiz*, and *Progress*. Figure 7 shows the resource list for each of the three course.

We aggregate the resource specific learning rates per *action type*, and plot radar graphs for three of the datasets in Figure 8(a, b and c). Resources as Video, Discussion and Wiki have the most impact on course **ColWri**; resources as Discussion, Wiki and Problem views have the most impact on course **CS**; and resources as Wiki and Discussion have the most impact on course **Stats**.

6. Conclusion

We present an open sourced tool (*xBKT*), which uses multicore computing and optimized c++ implementation, to solve the computational tractability issue of scaling student models to massive open online environments. We simplify the knowledge representations of online courses by automatically generating KC models from courseware structures at different level of conceptual granularity. We propose an extension of the Bayesian Knowledge Tracing by incorporating the students' activities on the platform and measure the impact of those activities on knowledge gained.

Our BKT models that included knowledge state trasition probabilities for each resource significantly improved predictive performance over a model that only looked at assessment information in all three datasets. This finding is of pedagogical significance as it may be indicative of estimated pedagogical efficacy of resources that generalizes to new students. Implications of this includes the ability to give model based feedback to instructors on the efficacy of various components of their course. Additionally, resources with high knowledge state transitions could be linked to as hints in the context of particular problems. With the contribution of this framework for measuring knowledge acquisition and performance enhancements, new frontiers in knowledge discovery in massive open online environments may be explored.

References

Beal, Matthew James. *Variational algorithms for approximate Bayesian inference*. PhD thesis, University of London, 2003.

Bernard, Joey. Running scientific code using ipython and scipy. *Linux Journal*, 2013(228):3, 2013.

Bezanson, Jeff, Karpinski, Stefan, Shah, Viral B., and Edelman, Alan. Julia: A fast dynamic language for technical computing. *CoRR*, abs/1209.5145, 2012.

Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, August 2006. ISBN 0387310738.

Chang, K., Beck, J., Mostow, J., and Corbett, A. A bayes net toolkit for student modeling in intelligent tutoring systems. *Proceedings of the 8th International Conference on Intelligent Tutoring Systems*, pp. 104–113, June 2006.

Corbett, Albert T and Anderson, John R. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4(4):253–278, 1994.

González-Brenes, JP, Huang, Yun, and Brusilovsky, Peter. General features in knowledge tracing: applications to multiple subskills, temporal item response theory, and expert knowledge. In *Proceedings of the 7th International Conference on Educational Data Mining (accepted, 2014)*, 2014.

Guennebaud, Gaël, Jacob, Benoît, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

Ho, A. D., Reich, J., Nesterko, S., Seaton, D. T., Mullaney, T., Waldo, J., and Chuang, I. Harvardx and mitx: The first year of open online courses (harvardx and mitx working paper no. 1). 2014.

Hoffman, Matthew D., Blei, David M., Wang, Chong, and Paisley, John. Stochastic variational inference. *Journal of Machine Learning Research*, 14:1303–1347, May 2013.

Initiative, ADL. Adl initiative, experience api. https://github.com/adlnet/- xAPI-Spec/blob/master/xAPI.md, 2013.

Intel Corporation. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. Number 248966-028. July 2013.

Johnson, Matthew J. and Willsky, Alan S. Bayesian nonparametric hidden semi-markov models. *Journal of Machine Learning Research*, 14:673–701, February 2013.

Lan, Andrew S, Studer, Christoph, and Baraniuk, Richard G. Time-varying learning and content analytics via sparse factor analysis. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 452–461. ACM, 2014.

Murphy, K. The bayes net toolbox for matlab. *Computing Science and Statistics*, pp. 33, 2001.

Pardos, Z. A., Bergner, Y., Seaton, D. T., and Pritchard, D. E. Adapting bayesian knowledge tracing to a massive open online course in edx. *Proceedings of the 6th International Conference on Educational Data Mining*, August 2013.

Rabiner, L. A tutorial on hmm and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2): 257–286, February 1989.

Xu, Y. and Mostow, J. Using logistic regression to trace multiple subskills in a dynamic bayes net. *Proceedings of the 4th International Conference on Educational Data Mining*, pp. 241–245, July 2011.

Table 3. RMSE per Resource scope per KC (Number of prior resources = *1*, and Resource granularity = *action*)

(a) **ColWri** course

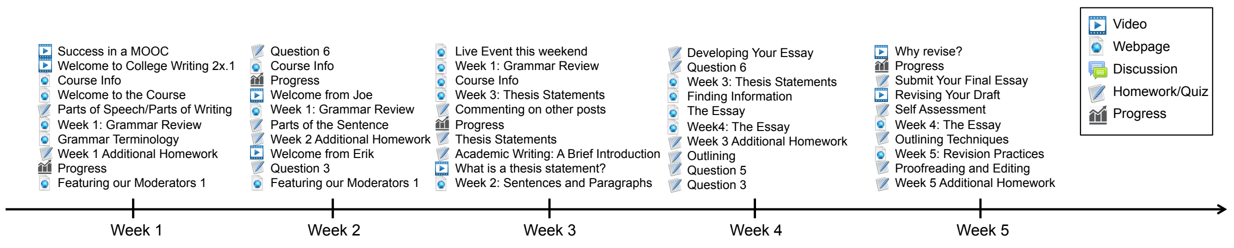
		KC Model					
		Subpart	Problem	Category	Category type	Chapter	Course
Resource scope:	Problem	*0.3289*	***0.3031***	*0.3071*	*0.3133*	*0.3135*	0.3171
	Category	0.3301	**0.3046**	0.3074	0.3138	0.3139	*0.3169*
	Chapter	0.3294	**0.3044**	0.3077	0.3136	0.3136	*0.3169*
	Course	0.3294	**0.3047**	0.3073	0.3136	*0.3135*	*0.3169*
	No resources (BKT)	0.3348	**0.3064**	0.3085	0.3140	0.3142	0.3172
Baseline		**0.3367**	0.3548	0.3556	0.3620	0.3662	0.3722

(b) **CS** course

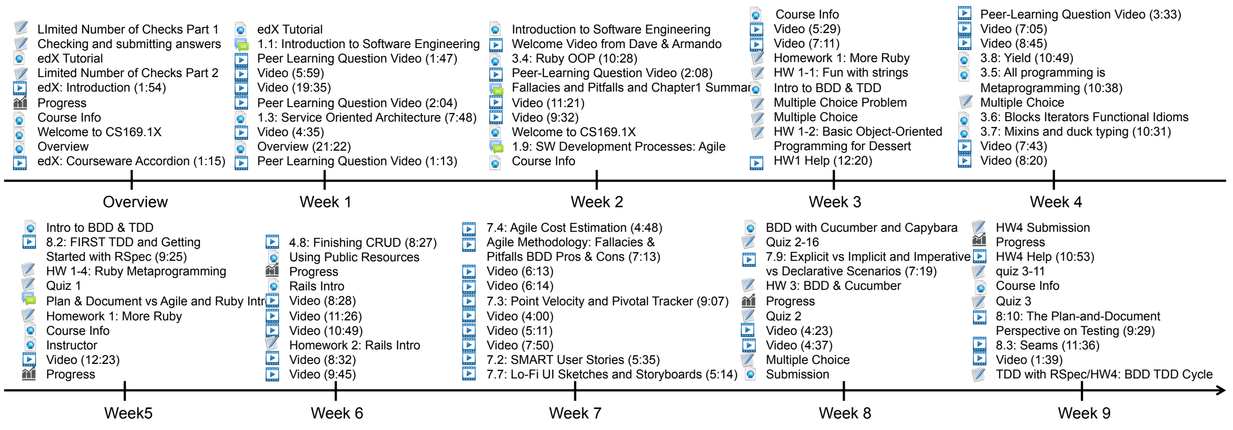
		KC Model					
		Subpart	Problem	Category	Category type	Chapter	Course
Resource scope:	Problem	*0.2169*	***0.2090***	*0.2132*	0.2143	*0.2158*	0.2176
	Category	0.2183	**0.2108**	0.2133	*0.2140*	0.2162	0.2176
	Chapter	0.2170	**0.2110**	0.2134	0.2141	0.2159	*0.2175*
	Course	0.2170	**0.2110**	0.2136	0.2141	0.2160	*0.2175*
	No resources (BKT)	0.2247	**0.2122**	0.2150	0.2156	0.2164	0.2176
Baseline		**0.2287**	0.2421	0.2486	0.2500	0.2517	0.2545

(c) **Stats** course

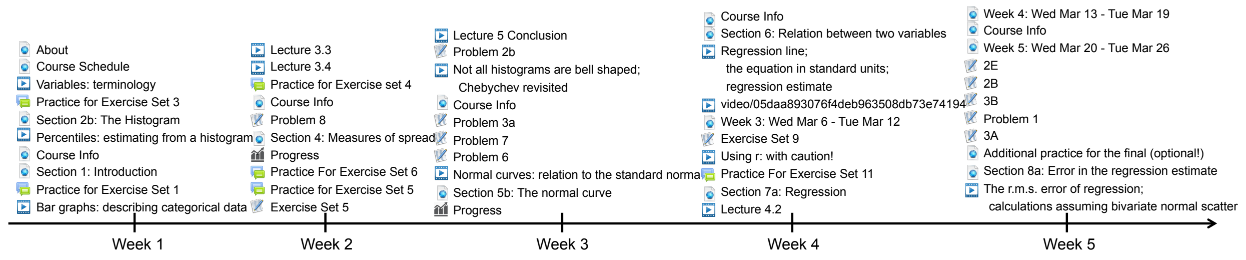
		KC Model					
		Subpart	Problem	Category	Category type	Chapter	Course
Resource scope:	Problem	*0.2491*	***0.2231***	*0.2362*	*0.2364*	*0.2432*	*0.2455*
	Category	0.2497	**0.2236**	0.2366	0.2365	*0.2432*	*0.2455*
	Chapter	0.2498	**0.2237**	0.2366	0.2368	*0.2432*	*0.2455*
	Course	0.2499	**0.2237**	0.2366	0.2369	*0.2432*	*0.2455*
	No resources (BKT)	0.2516	**0.2249**	0.2366	0.2366	*0.2432*	*0.2455*
Baseline		**0.2577**	0.2790	0.2851	0.2851	0.2892	0.2908



(a) **ColWri** course

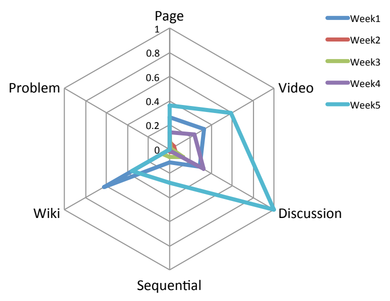

LImited Number of Checks Part 1
Checking and submitting answers
edX Tutorial
Limited Number of Checks Part 2
edX: Introduction (1:54)
Progress
Course Info
Welcome to CS169.1X
Overview
edX: Courseware Accordion (1:15)
edX Tutorial
1.1: Introduction to Software Engineering
Peer Learning Question Video (1:47)
Video (5:59)
Video (19:35)
Peer Learning Question Video (2:04)
1.3: Service Oriented Architecture (7:48)
Video (4:35)
Overview (21:22)
Peer Learning Question Video (1:13)
Introduction to Software Engineering
Welcome Video from Dave & Armando
3.4: Ruby OOP (10:28)
Peer-Learning Question Video (2:08)
Fallacies and Pitfalls and Chapter1 Summar
Video (11:21)
Video (9:32)
Welcome to CS169.1X
1.9: SW Development Processes: Agile
Course Info
Course Info
Video (5:29)
Video (7:11)
Homework 1: More Ruby
HW 1-1: Fun with strings
Intro to BDD & TDD
Multiple Choice Problem
Multiple Choice
HW 1-2: Basic Object-Oriented Programming for Dessert
HW1 Help (12:20)
Peer-Learning Question Video (3:33)
Video (7:05)
Video (8:45)
3.8: Yield (10:49)
3.5: All programming is Metaprogramming (10:38)
Multiple Choice
3.6: Blocks Iterators Functional Idioms
3.7: Mixins and duck typing (10:31)
Video (7:43)
Video (8:20)
Overview
Week 1
Week 2
Week 3
Week 4
Intro to BDD & TDD
8.2: FIRST TDD and Getting Started with RSpec (9:25)
HW 1-4: Ruby Metaprogramming
Quiz 1
Plan & Document vs Agile and Ruby Intr
Homework 1: More Ruby
Course Info
Instructor
Video (12:23)
Progress
4.8: Finishing CRUD (8:27)
Using Public Resources
Progress
Rails Intro
Video (8:28)
Video (11:26)
Video (10:49)
Homework 2: Rails Intro
Video (8:32)
Video (9:45)
7.4: Agile Cost Estimation (4:48)
Agile Methodology: Fallacies & Pitfalls BDD Pros & Cons (7:13)
Video (6:13)
Video (6:14)
7.3: Point Velocity and Pivotal Tracker (9:07)
Video (4:00)
Video (5:11)
Video (7:50)
7.2: SMART User Stories (5:35)
7.7: Lo-Fi UI Sketches and Storyboards (5:14)
BDD with Cucumber and Capybara
Quiz 2-16
7.9: Explicit vs Implicit and Imperative vs Declarative Scenarios (7:19)
HW 3: BDD & Cucumber
Progress
Quiz 2
Video (4:23)
Video (4:37)
Multiple Choice
Submission
HW4 Submission
Progress
HW4 Help (10:53)
quiz 3-11
Course Info
Quiz 3
8:10: The Plan-and-Document Perspective on Testing (9:29)
8.3: Seams (11:36)
Video (1:39)
TDD with RSpec/HW4: BDD TDD Cycle
Week5
Week 6
Week 7
Week 8
Week 9

(b) **CS** course

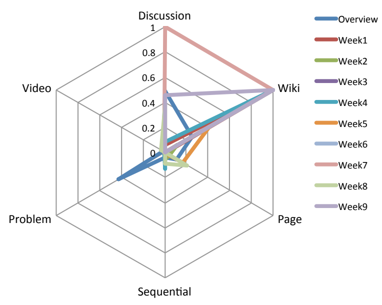

About
Course Schedule
Variables: terminology
Practice for Exercise Set 3
Section 2b: The Histogram
Percentiles: estimating from a histogram
Course Info
Section 1: Introduction
Practice for Exercise Set 1
Bar graphs: describing categorical data
Lecture 3.3
Lecture 3.4
Practice for Exercise set 4
Course Info
Problem 8
Section 4: Measures of spread
Progress
Practice for Exercise Set 6
Practice for Exercise Set 5
Exercise Set 5
Lecture 5 Conclusion
Problem 2b
Not all histograms are bell shaped; Chebychev revisited
Course Info
Problem 3a
Problem 7
Problem 6
Normal curves: relation to the standard norma
Section 5b: The normal curve
Progress
Course Info
Section 6: Relation between two variables
Regression line; the equation in standard units; regression estimate
video/05daa893076f4deb963508db73e74194
Week 3: Wed Mar 6 - Tue Mar 12
Exercise Set 9
Using r: with caution!
Practice For Exercise Set 11
Section 7a: Regression
Lecture 4.2
Week 4: Wed Mar 13 - Tue Mar 19
Course Info
Week 5: Wed Mar 20 - Tue Mar 26
2E
2B
3B
Problem 1
3A
Additional practice for the final (optional!)
Section 8a: Error in the regression estimate
The r.m.s. error of regression; calculations assuming bivariate normal scatter
Week 1
Week 2
Week 3
Week 4
Week 5

(c) **Stats** course

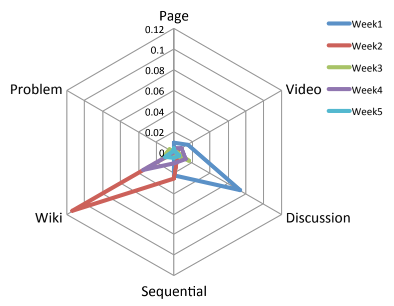
Figure 7. Top 10 recommended resources across weeks (Number of prior resources = *5*, Resource granularity = *object*, KC model = *chapter*, and Resource scope = *course*)



(a) **ColWri** course



(b) **CS** course



(c) **Stats** course

Figure 8. Impact of resource type on learning rates over weeks