From Predictive Models to Instructional Policies

Joseph Rollinson

Advisor: Emma Brunskill

May 1, 2015

Abstract

Intelligent tutoring systems provide their students with an adaptive personalized learning experience. To do so, intelligent tutoring systems attempt to capture the state of their students through a student model. Student models have two primary uses: prediction of future student performance and instructional decision making. Since prediction performance is easier to quantify, student models are frequently judged by their predictive power. This has bred student models that are very powerful predictors, but cannot be easily used in decision making. In this work, we leverage these powerful predictors using novel decision algorithms that are compatible with almost any predictive student model. In particular we consider two decision problems: when to stop providing questions to the student and which skill to practice next. Our results suggest that our when-to-stop decision algorithm acts similarly on existing decision algorithms with the added benefit of stopping when students are unable to progress given the current material. Our preliminary work on deciding between skills suggests that logistic regression models, previously only used for prediction, can be used to pick between skills and even learn a skill hierarchy.

Contents

Chapter 1

Introduction

Education is an incredibly important factor in human wellbeing. It is connected to both economic and social improvement. For example in the United States, every additional year of education boosts average annual earnings by 7–10%, and studies in the United Kingdom showed that more schooling also boosts voter turnout [17]. Improving access to education is so important that the U.N. made achieving universal primary education the second Millennium Development Goal. However, the UNESCO Center for Statistics reported that nearly 58 million children of primary school age were not enrolled in school in 2012. Thus, improving the ability to access education is still vitally important.

One-to-one tutoring can have a huge impact on learning. Most students are taught using conventional techniques such as classrooms or lecture halls. Bloom found that students who were tutored one-to-one performed on average better than 98% of students taught using conventional methods [3]. Unfortunately, tutoring using instructors is prohibitively expensive for most people. However, this does suggest that there are drastic improvements that can be made to the education systems of even the most developed countries.

Intelligent tutoring systems provide a personalized education to their users. Unlike classes, intelligent tutoring systems do not teach all students in the same way. Instead, they adapt their instruction based on the student's performance. In this regard, they have similar characteristics to human one-to-one tutors. The performance of intelligent tutoring systems is only limited by the algorithms that they are built upon, and we believe that there is a large opportunity to improve their quality. Since intelligent tutoring systems are programs, they are scalable. As the cost of technology declines, so too should the cost of intelligent tutoring systems. Improving intelligent tutoring systems could have an incredible impact both on the quality and availability of education globally.

In this thesis, we are going to focus on a simplified intelligent tutoring system model that gives students questions relating to specific skills and reacts to the correctness of the student's answer, as shown in figure 1.1. Although quite simple, these models can be clearly reasoned about. Also, many real-world intelligent tutoring systems are based upon this model. As shown in figure 1.2, intelligent tutoring systems consist of a student model and instructional policies. Student models are responsible for maintaining a representation of the student being taught. Instructional policies are responsible for making instructional decisions such as which skill to teach next or when to stop teaching the student.

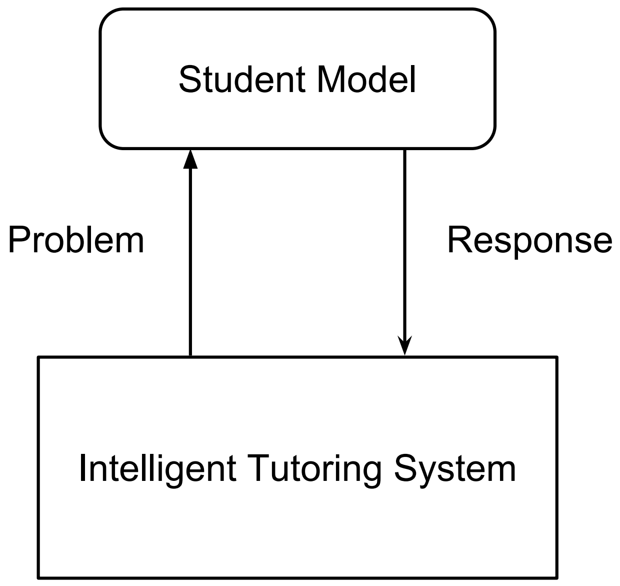


Figure 1.1: A simplified intelligent tutoring system model. The intelligent tutoring system provides the student with problems and makes decisions according to the correctness of the student's responses.

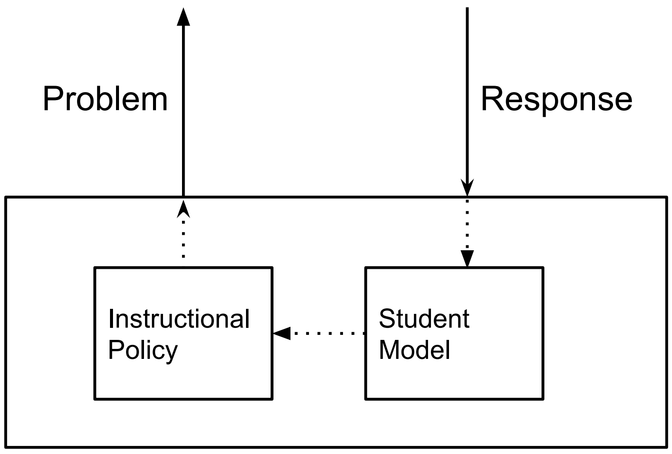


Figure 1.2: Intelligent tutoring system internals. The student model reacts to student responses, and the instructional policy uses the student model to make decisions.

There has been significant interest in cognitive models used within activity design (often referred to as the inner-loop) and even authoring tools developed to make designing effective activities easier (e.g. CTAT [1]). However, there has been much less attention to outer-loop (what problem to select or when to stop) instructional policies (though exceptions include [7, 15, 21]).

Within the ITS research community, there has been substantial work on constructing student models that can accurately predict student performance (e.g. [8, 5, 19, 7, 13, 12, 18, 9]). In fact, the 2010 KDD Cup challenge was to predict student performance in data collected from students using the Carnegie Learning's Cognitive Tutor [22]. Unfortunately, many predictive student models cannot be used by any instructional policy. This means that much of the work done in predictive student models cannot be used to build better intelligent tutoring systems.

In this work, we attempt to bring these models back into the fold by providing instructional policies that can use any predictive student model. We introduce a predictive student model function interface that allows us to abstract out the underlying student models from the instructional policies. This allows us to modify the policy or student model without affecting our implementation of the other. We find that model agnostic instructional policies also provide a new method of comparing student models. Our results suggest that instructional policies based on student models with similar predictive accuracies can make very different decisions. Comparing the decisions made by policies based on student models provides insight into the differences between the models and shows possible areas for improvement.

In particular, the contributions of this thesis are:

- A model agnostic when-to-stop policy that stops both when students master a skill and when students are unable to master a skill with the given instruction.
- A modified Performance Factors Model that prevents extreme prediction asymptotes through windowing.
- Two model agnostic skill-choice policies that can be used with respect to any goal.

We now describe each of these contributions before finishing with future work and conclusions.

Chapter 2

The When-To-Stop Problem

This chapter will be published at the 2015 EDM conference under the title "From Predictive Models to Instructional Policies".

2.1 Introduction

In this chapter we focus on a common outer-loop ITS challenge, adaptively deciding when to stop teaching a certain skill to a student given correct/incorrect responses. Somewhat surprisingly, there are no standard policy rules or algorithms for deciding when to stop teaching for many of the student models introduced over the last decade. Bayesian Knowledge Tracing [8] naturally lends itself to mastery teaching, since one can halt when the student has mastered a skill with probability above a certain threshold. Such a mastery threshold has been used as part of widely used tutoring systems, but typically in conjunction with additional rules since a student may never reach a sufficient mastery threshold given the available activities.

We seek to be able to directly use a wide range of student models to create instructional policies that halt both when a student has learned a skill and when the student seems unlikely to make any further progress given the available tutoring activities. To do so we introduce an instructional policy rule based on change in predicted student performance.

Our specific contributions are as follows:

- We provide a functional interface to student models that captures their predictive powers without knowledge of their internal mechanics (Section 2.3).
- We introduce the *predictive similarity policy*: a new when-to-stop policy that can take as input any predictive student-model (Section 2.4) and can halt both if students have successfully acquired a skill or do not seem able to do so given the available activities.
- We analyze the performance of this policy compared to a mastery threshold policy on the KDD dataset and find our policy tends to suggest similar or a smaller number of problems than a mastery threshold policy (Section 2.5).
- We also show that our new policy can be used to analyze a range of student models with similar predictive performance (on the KDD dataset) and find that they can sometimes suggest very different numbers of instructional problems. (Section 2.5).

Our results suggest that predictive accuracy alone can mask some of the substantial differences among student models. Polices based on models with similar predictive accuracy can make widely different decisions. One direction for future work is to measure which models produce the best learning policies. This will require new experiments and datasets.

2.2 Background: Student Models

Student models are responsible for modeling the learning process of students. The majority of student models are *predictive models* that provide probabilistic predictions of whether a student will get a subsequent item correct. In this section we describe two popular predictive student models, *Bayesian knowledge tracing* and *latent factor models*. Note that other predictive models, such as Predictive State Representations (PSRs), can also be used to calculate the probability of a correct response [9].

2.2.1 Bayesian Knowledge Tracing

Bayesian Knowledge Tracing (BKT) [8] tracks the state of the student's knowledge as they respond to practice questions. BKT treats students as being in one of two possible hidden states: *Mastery* and *Non-Mastery*. It is assumed that a student never forgets what they have mastered and if not yet mastered, a new question always has a fixed static probability of helping the student master the skill. These assumptions mean that BKT requires only four trained parameters:

$P(L_0)$	Initial probability of mastery.
$P(T)$	Probability of *transitioning* to mastery over a single learning opportunity.
$P(G)$	Probability of *guessing* the correct answer when the student is not in the mastered state.
$P(S)$	Probability of *slipping* (making a mistake) when the student is in the mastered state.

After every response, the probability of mastery, $P(L_t)$, is updated with Bayesian inference. The probability that a student responds correctly is

$$P_{\text{BKT}}(C_t) = (1 - P(S))P(L_t) + P(G)(1 - P(L_t)). \tag{2.1}$$

Prior work suggests that students can get stuck on a particular activity. Unfortunately, BKT as described above assumes that students will inevitably master a skill if given enough questions. As this is not always the case, in industry BKT is often used together with additional rules to make instructional decisions.

2.2.2 Latent Factor Models

Unlike BKT models, Latent Factor Models (LFM) do not directly model learning as a process [5]. Instead, LFMs assume that there are latent parameters of both the student and skill that can be used to predict student performance. These parameters are learned from a dataset of students answering questions on multiple skills. The probability that the student responds correctly to the next question is calculated by applying the sigmoid function to the linear combination of parameters p and

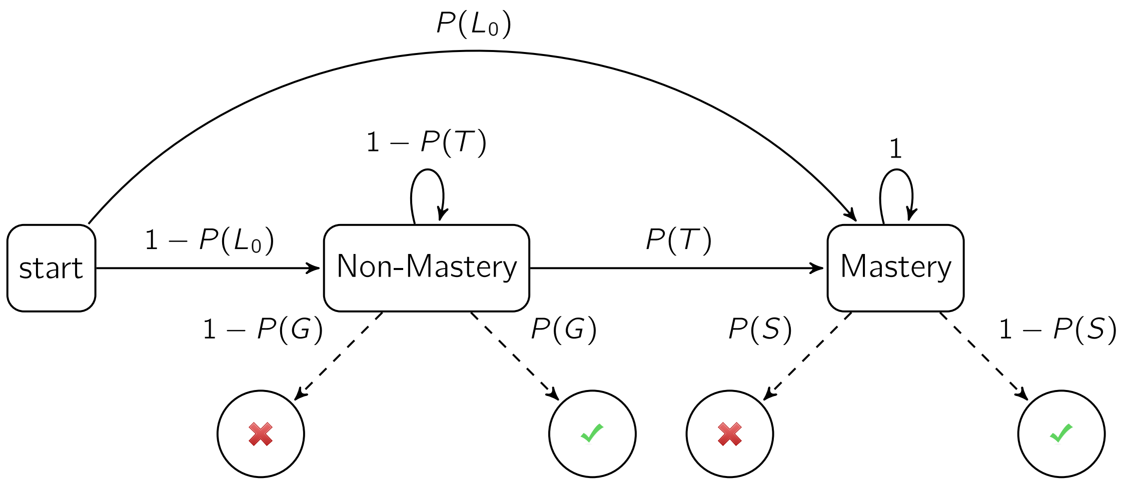


Figure 2.1: BKT as a Markov process. *Mastery* and *Non-Mastery* are hidden states. Arrow values represent the probability of the transition or observation.

features f.

$$P_{\text{LFM}}(C) = \frac{1}{1 - e^{-f \cdot p}} \tag{2.2}$$

Additive Factor Models (AFM) [5] are based on the assumption that student performance increases with more questions. A student is represented by an aptitude parameter (α_i) and a skill is represented by a difficulty parameter (β_k) and learning rate (γ_k). AFM is sensitive to the number of questions the student has seen, but ignores the correctness of student responses. The probability that student i will respond correctly after n responses on skill k is

$$P_{\text{AFM}}(C) = \frac{1}{1 - e^{-(\alpha_i + \beta_k + \gamma_k n)}}. \tag{2.3}$$

Performance Factor Models (PFM) [19] are an extension of AFMs that are sensitive to the correctness of student responses. PFMs separate the skill learning rate into success and failure parameters, μ_k and ρ_k respectively. The probability that student i will respond correctly after s correct responses and f incorrect responses on skill k is

$$P_{\text{PFM}}(C) = \frac{1}{1 - e^{-(\alpha_i + \beta_k + \mu_k s + \rho_k f)}}. \tag{2.4}$$

LFMs can easily be extended to capture other features. For example, the instructional factors model [7] extends PFMs with a parameter for the number of tells (interactions that do not generate observations) given to the student. To our knowledge there is almost no work on using LFMs to capture temporal information about the order of observations. Unlike BKT, LFMs are not frequently used in instructional policies.

Though structurally different, BKT models, AFMs and PFMs tend to have similar predictive accuracy [12, 19]. This raises the interesting question of whether instructional policies that use these models are similar.

2.3 When-To-Stop Policies

We assume a simple intelligent tutoring system that teaches students one skill at a time. All questions are treated the same, so the system only has to decide when to stop providing the student questions. In this section, we provide a general framework for the when-to-stop problem. In particular, we describe an interface that abstracts out the student model from instructional policies, which we will use to define the MASTERY THRESHOLD policy and use in the next section as the foundations of a model-agnostic instructional policy.

2.3.1 Accessing Models

Policies require a mechanism for getting values from student models to make decisions. We describe this mechanism as a state type and a set of functions. A student model consists of two types of values: immutable parameters that are learned on training data and mutable state that changes over time. For example, the parameters for BKT are $(P(L_0), P(T), P(G), P(S))$ and the model state is the probability of mastery ($P(L_t)$). Policies treat the state as a black box, which they pass to functions. All predictive student models must provide the following functions. **startState**$(\dots)$ returns the model state given that the student has not seen any questions. **updateState**$(\text{state}, \text{obs})$ returns an updated state given the observation. For this chapter, observations are whether the stu-

Table 2.1: Functional interfaces for BKT and PFM

	BKT	PFM
startState$(\dots)$	$P(L_0)$	$(\alpha_i + \beta_k, \mu_k, \rho_k, 0, 0)$
updateState(s, o)	$P(L_{t+1} \vert P(L_t), O_{t+1} = o)$	$\begin{cases}(w, \mu, \rho, s+1, f) & \text{if } o = C \\ (w, \mu, \rho, s, f+1) & \text{if } o = \neg C\end{cases}$
predictCorrect(s)	$P(\neg S)P(L_t) + P(G)(1 - P(L_t))$	$\left(1 + e^{-(w+s\mu+f\rho)}\right)^{-1}$
predictMastery(s)	$P(L_t)$	—

dent got the last question correct or incorrect. Finally, predictive student models must provide **predictCorrect**(state), which returns the probability that the student will get the next question correct. The function interfaces for BKT models and PFM are provided in table 2.1. Under this abstraction, when-to-stop policies are functions **stop**(state) that output true if the system should stop providing questions for the current skill and false if the system should continue providing the student with questions.

2.3.2 Mastery Threshold Policy

The MASTERY THRESHOLD POLICY halts when the student model is confident that the student has mastered the skill. This implies that we want to halt when the student masters the skill. Note that if the estimate of student mastery is based solely on a BKT[1] then a mastery threshold policy implicitly assumes that every student will master the skill given enough problems. Mathematically, we want to stop at time t if $P(L_t) > \Delta$, where Δ is our mastery threshold. The MASTERY THRESHOLD policy function can be written as:

$$\textbf{stop}_M(\text{state}) = \textbf{predictMastery}(\text{state}) > \Delta. \tag{2.5}$$

[1]In practice, industry systems that use mastery thresholds and BKTs often use additional rules as well.

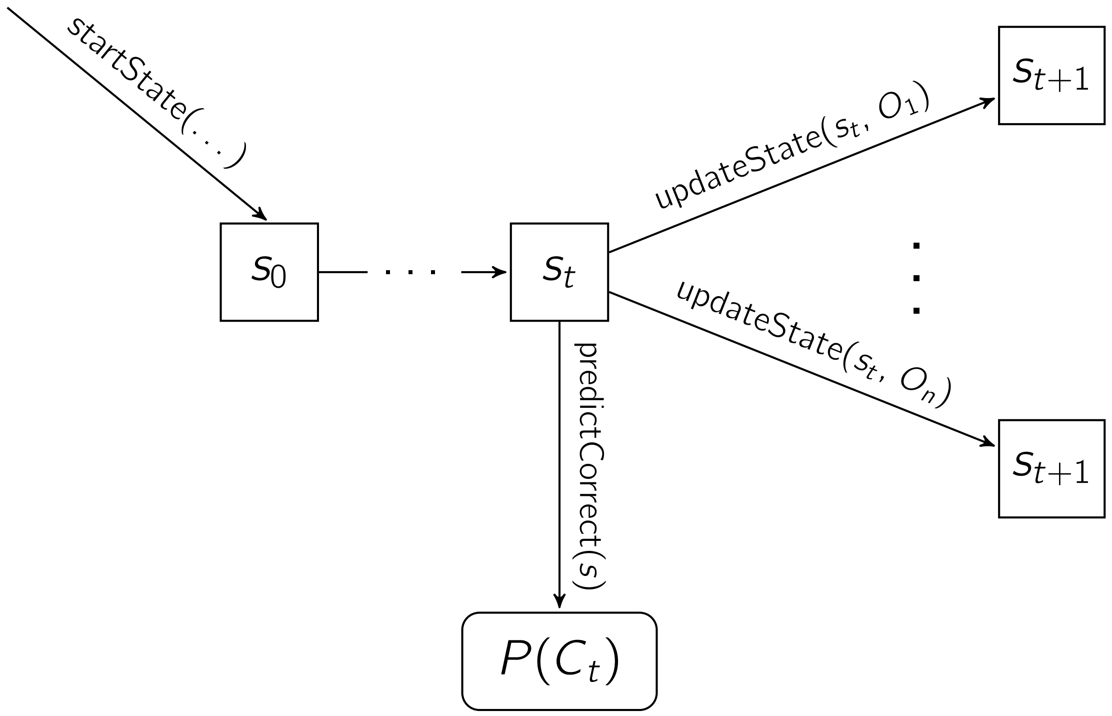


Figure 2.2: Model process with functional interface

The MASTERY THRESHOLD policy can only be used with models that include **predictMastery**(state) in their function set. BKT models are compatible, but LFMs are not. By itself, the MASTERY THRESHOLD does not stop if the student has no chance of attaining mastery in the skill with the given activities. Students on poorly designed skills could be stuck learning a skill indefinitely.

2.4 From Prediction to Policy

In educational data mining, a large emphasis is put on building models that can accurately predict student observations. Our goal is to build a new when-to-stop policy that will work with any predictive student model.

Our new instructional policy is based on a set of assumptions. First, students working on a skill will eventually end in one of two hidden end-states. Either, they will master the skill, or they will be unable to master the skill given the activities available. Second, once students enter either end-state, the probability that they respond correctly to a question stays the same. Third, if the probability that a student will respond correctly is not changing, then the student is in an end-state. Finally, we should stop if the student is in an end-state.

From these assumptions it follows that if the probability that the student will respond correctly to the next question is not changing, then we should stop. In other words, we should stop if it is highly likely that showing the student another question will not change the probability that the student will get the next question correct by a significant amount. We propose to stop if

$$(P(|P(C_t) - P(C_{t+1})| < \epsilon)) > \delta \tag{2.6}$$

where $P(C_t)$ is the probability that the student will get the next question right. This can be thought of as a threshold on the sum of the probabilities of each observation that will lead to an insignificant change in the probability that a student will get the next question correct, which can be written as

$$\sum_{o \in O} P(O_t = o)\mathbb{1}(|P(C_t) - P(C_{t+1}|O_t = o)| < \epsilon) > \delta \tag{2.7}$$

where $P(C_{t+1}|O_t = o)$ is the probability that the student will respond correctly after observation o, O_t is the observation at time t, and $\mathbb{1}$ is an indicator variable. In our case $O = \{C, \neg C\}$. This expression is true in the following cases:

1. $P(C_t) > \delta$ and $|P(C_t) - P(C_{t+1}|C_t)| < \epsilon$
2. $P(\neg C_t) > \delta$ and $|P(C_t) - P(C_{t+1}|\neg C_t)| < \epsilon$
3. $|P(C_t) - P(C_{t+1}|C_t)| < \epsilon$ and
   $|P(C_t) - P(C_{t+1}|\neg C_t)| < \epsilon$

First, if a student is highly likely to respond correctly to the next question and the change in prediction is small if the student responds correctly, then we should stop. Second, if a student is highly unlikely to respond correctly to the next question and the change in prediction is small if the student responds incorrectly, then we should stop. Third, if the change in prediction is small no matter how the student responds, then we should stop. All terms in these expressions can be calculated from the predictive student model interface as shown in equations 2.8 and 2.9. We call the instructional policy that stops according to these three cases the PREDICTIVE SIMILARITY policy. The function for the PREDICTIVE SIMILARITY policy is provided in algorithm 1

$$P(C_t) = \textbf{predictCorrect}(s) \tag{2.8}$$

$$P(C_{t+1}|O_t) = \textbf{predictCorrect}(\textbf{updateState}(s, O_t)) \tag{2.9}$$

Algorithm 1 PREDICTIVE SIMILARITY policy stop function

```
1:  function STOP(state)
2:      $P(C_t) \leftarrow$ predictCorrect(state)
3:      total $\leftarrow 0$
4:      if $P(C_t) > 0$ then
5:          state' $\leftarrow$ updateState(state, correct)
6:          $P(C_{t+1}|C_t) \leftarrow$ predictCorrect(state')
7:          if $|P(C_t) - P(C_{t+1}|C_t)| < \epsilon$ then
8:              total $\leftarrow$ total $+ P(C_t)$
9:      if $P(C_t) < 1$ then
10:         state' $\leftarrow$ updateState(state, incorrect)
11:         $P(C_{t+1}|\neg C_t) \leftarrow$ predictCorrect(state')
12:         if $|P(C_t) - P(C_{t+1}|\neg C_t)| < \epsilon$ then
13:             total $\leftarrow$ total $+ (1 - P(C_t))$
14:     return total $> \delta$
```

2.5 Experiments & Results

We now compare the PREDICTIVE SIMILARITY policy to the MASTERY THRESHOLD policy and see if using different student models as input to the predictive SIMILARITY POLICY yields quantitatively different policies.

2.5.1 ExpOps

In order to better understand the differences between two instructional policies we will measure the expected number of problems to be given to students by a policy using the ExpOps algorithm. The ExpOps algorithm allows us to summarize an instructional policy into a single number by approximately calculating the expected number of questions an instructional policy would provide to a student. A naive algorithm takes in the state of the student model and returns 0 if the instructional policy stops at the current state or recursively calls itself with an updated state given each possible observation as shown in equation 2.10. It builds a synthetic tree of possible observations and their probability using the model state. The tree grows until the policy decides to stop teaching the student. This approach does not require any student data nor does it generate any observation sequences. However, this algorithm may never stop, so ExpOps approximates it by also stopping if we reach a maximum length or if the probability of the sequence of observations thus far drops below a path threshold as shown in algorithm 2. In this chapter, we use a path threshold of 10^{-7} and a maximum length of 100.

$$E[Ops] = \begin{cases} 0 & \text{if } \textbf{stop}(s) \\ 1 + \sum\limits_{o \in O} P(O_t = o) E[Ops|o] & \text{otherwise} \end{cases} \tag{2.10}$$

Lee and Brunskill first introduced this metric to show that individualized models lead to significantly different policies than the general models [15].

2.5.2 Data

For our experiments we used the Algebra I 2008–2009 dataset from the KDD Cup 2010 Educational Data Mining Challenge [22]. This dataset was collected from students learning algebra I using Carnegie Learning Inc.'s intelligent tutoring systems. The dataset consists of 8,918,054 rows where each row corresponds to a single step inside a problem. These steps are tagged according to three different knowledge component models. For this chapter, we used the SubSkills knowl-

Algorithm 2 Expected Number of Learning Opportunites

```
function ExpOps(startState)
    function ExpOps'(state, $P$(path), len)
        if $P$(path) $<$ pathThreshold then
            return $0$
        if len $\geq$ maxLen then
            return $0$
        if stop(state) then
            return $0$
        $P(C) \leftarrow$ predictCorrect(state)
        $P(W) \leftarrow 1 - P(C)$
        expOpsSoFar $\leftarrow 0$
        if $P(C) > 0$ then
            $P(\text{path} + \text{c}) \leftarrow P(\text{path}) * P(C)$
            $\text{state}' \leftarrow \text{updateState}(\text{state}, C)$
            $\text{ops} \leftarrow$ ExpOps'$(\text{state}', P(\text{path} + \text{c}), \text{len} + 1)$
            $\text{expOpsSoFar} \leftarrow \text{expOpsSoFar} + (\text{ops} * P(C))$
        if $P(W) > 0$ then
            $P(\text{path} + \text{w}) \leftarrow P(\text{path}) * P(W)$
            $\text{state}' \leftarrow \text{updateState}(\text{state}, \text{incorrect})$
            $\text{ops} \leftarrow$ ExpOps'$(\text{state}', P(\text{path} + \text{w}), \text{len} + 1)$
            $\text{expOpsSoFar} \leftarrow \text{expOpsSoFar} + (\text{ops} * P(W))$
        return 1 + expOpsSoFar
    return ExpOps'((startState, 1, 0))
```

edge component model. We removed all rows with missing data. We combined the rows into observation sequences per student and per skill. Steps attached to multiple skills were added to the observation sequences of all attached skills. We removed all skills that had less than 50 observation sequences. Our final dataset included 3292 students, 505 skills, and 421,991 observation sequences.

We performed 5-fold cross-validation on the datasets to see how well AFM, PFM, and BKT models predict student performance. We randomly separated the dataset into five folds with an equal number of observation sequences per skill in each fold. We trained AFM, PFM, and BKT models on four of the five folds and then predicted student performance on the leftover fold. We calculated the root mean squared error found in Table 2.2. Our results show that the three models had similar predictive accuracy, agreeing with prior work.

Table 2.2: Root Mean Squared Error on 5 Folds

Fold	BKT	PFM	AFM
0	0.353	0.364	0.368
1	0.359	0.367	0.371
2	0.358	0.368	0.371
3	0.366	0.369	0.374
4	0.353	0.365	0.368

2.5.3 Model Implementation

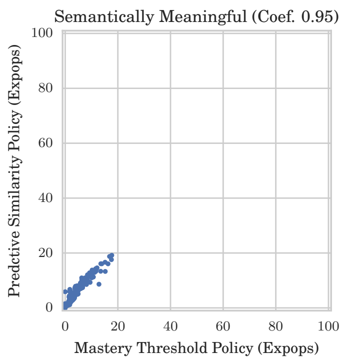
We implemented BKT models as hidden Markov models using a python package we developed. We used the Baum-Welch algorithm to train the models, stopping when the change in log-likelihood between iterations fell below 10^{-5}. For each skill, 10 models with random starting parameters were trained, and the one with the highest likelihood was picked. Both AFM and PFM were implemented using scikit-learn's logistic regression classifier [20]. We used L1 normalization and included a fit intercept. The tolerance was 10^{-4}. We treated an observation connected to multiple skills as multiple observations, one per skill. It is also popular to treat them as a single observation with multiple skill parameters. In the interest of reproducibility, we have published the models used as a python package.[2]

2.5.4 Experiment 1: Comparing policies

The MASTERY THRESHOLD policy is frequently used as a key part of deciding when to stop showing students questions. However without additional rules, it does not stop if students cannot learn the skill from the current activities. In this experiment we compare the PREDICTIVE SIMILARITY policy to the MASTERY THRESHOLD policy to see if the PREDICTIVE SIMILARITY policy acts like the MASTERY THRESHOLD policy when students learn and stops sooner when students are unable to learn with the given tutoring. We based both policies on BKT models.

We ran ExpOps on each skill for both policies. For the MASTERY THRESHOLD policy, we

[2]The packages are available at `http://www.jrollinson.com/research/2015/edm/from-predictive-models-to-instructional-policies.html`.



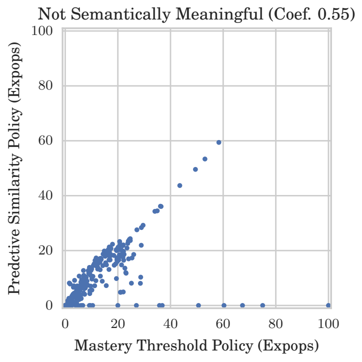


Figure 2.3: ExpOps using the MASTERY THRESHOLD policy and the PREDICTIVE SIMILARITY policy on skills with and without semantically meaningful parameters.

used the community standard threshold of $\Delta = 0.95$. For the PREDICTIVE SIMILARITY policy, we decided that the smallest meaningful change in predictions is 0.01 and that our confidence should be 0.95, so we set $\epsilon = 0.01$ and $\delta = 0.95$. We then split the skills into those where the BKT model trained on them had semantically meaningful parameters and the rest. A BKT model was said to have semantically meaningful parameters if $P(G) \leq 0.5$ and $P(S) \leq 0.5$. 218 skills had semantically meaningful parameters and 283 did not.[3].

The Pearson correlation coefficient between ExpOps values calculated using the two policies on skills with semantically meaningful parameters was 0.95. This suggests that the two policies make very similar decisions when based on BKT models with semantically meaningful parameters. However, the Pearson correlation coefficient between ExpOps values calculated using the two policies on skills that do not have semantically meaningful parameters was only 0.55. To uncover

[3]We found similar results for both experiments using BKT models trained through brute force iteration on semantically meaningful values. These results can be found at `http://www.jrollinson.com/research/2015/edm/from-predictive-models-to-instructional-policies.html`

why the correlation coefficient was so much lower on skills that do not have semantically meaningful parameters, we plotted the ExpOps values calculated with the MASTERY THRESHOLD policy on the X-axis and the ExpOps values calculated with the PREDICTIVE SIMILARITY policy on the Y-axis for each skill as shown in figure 2.3. This plot shows that the PREDICTIVE SIMILARITY policy tends to either agree with the MASTERY THRESHOLD policy or have a lower ExpOps value on skills with parameters that are not semantically meaningful. This suggests that the PREDICTIVE SIMILARITY policy is stopping sooner on skills that students are unlikely to learn. The mastery policy does not give up on these skills, and instead teaches them for a long time.

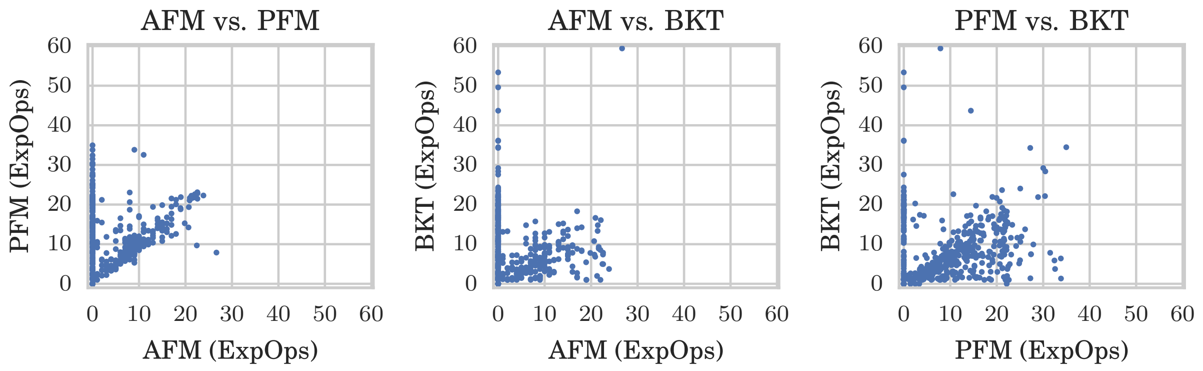


Figure 2.4: ExpOps plots for the PREDICTIVE SIMILARITY policy using BKT, AFM, and PFM.

2.5.5 Experiment 2: Comparing models with the predictive similarity policy

The previous experiment suggests that the PREDICTIVE SIMILARITY policy can effectively mimic the good aspects MASTERY THRESHOLD policy when based on a BKT model. We now wish to see how using models with similar predictive accuracy, but different internal structure will affect it. LFMs and BKT models have vastly different structure making them good models for this task. Our earlier results also found that AFM, PFM, and BKT models have similar predictive accuracy. We ran ExpOps on each skill with the PREDICTIVE SIMILARITY policy based both on AFM and PFM. AFM and PFM require a student parameter, which we set to the mean of their trained student parameters. This is commonly done when modeling a student that has not been seen before. We

Table 2.3: Correlation coefficients on ExpOps values from policies using BKT, AFM, and PFM.

Models	Coefficient with all skills	Coefficient with skills not stopped immediately
AFM vs. PFM	0.32	0.72
AFM vs. BKT	-0.06	0.44
PFM vs. BKT	0.16	0.46

compared the ExpOps values for these two models with the values for the BKT-based PREDICTIVE SIMILARITY policy calculated in the previous experiment.

We first looked at how many skills the different policies immediately stopped on. We found that the BKT-based policy stopped immediately on 31 (6%) of the skills, whilst PFM stopped immediately on 130 (26%) and AFM stopped immediately on 295 (59%).

We calculated the correlation coefficient between each pair of policies on all skills as well as just on skills in which both policies did not stop immediately as shown in table 2.3. We found that AFM and PFM had the highest correlation coefficient. For each pair of policies, we found that removing the immediately stopped skills had a large positive impact on correlation coefficient. The BKT-based policy had a correlation coefficient of 0.44 with the AFM-based policy and 0.46 with the PFM-based policy on skills that were not immediately stopped on. This suggests that there is a weak correlation between LFM-based and BKT-based policies.

We plotted the ExpOps values for each pair of policies, shown in figure 2.4. The AFM vs. PFM plot reiterates that the AFM-based and PFM-based policies have similar ExpOps values on skills where AFM does not stop immediately. The BKT vs. PFM plot shows that the PFM-based policy either immediately stops or has a higher ExpOps value than the BKT-based policy on most skills.

To understand why the PFM-based policy tends to either stop immediately or go on for longer than the BKT-based policy, we studied two skills. The first skill is ‘Plot point on minor tick mark — integer major fractional minor’ on which the BKT-based policy has an ExpOps value of 7.0 and the PFM-based policy has an ExpOps value of 20.7. The second skill is ‘Identify solution type of compound inequality using and’ on which the BKT-based policy has an ExpOps value of 11.4 and the PFM-based policy immediately stops. We calculated the predictions of both models on two artificial students, one who gets every question correct and one who gets every question incorrect. In figure 2.5, we plot the prediction trajectories to see how the predictions of the two models compare. In both plots, the PFM-based policy asymptotes slower than the BKT-based policy. Since LFMs calculate predictions with a logistic function, PFM predictions asymptote to 0 when given only incorrect responses and 1 when given only correct responses, whereas the BKT model's predictions asymptote to $P(G)$ and $1 - P(S)$ respectively. In the first plot, the PFM-based policy learns at a slower rate than the BKT-based policy, but the predictions do begin to asymptote by the 20th question. In the second plot, the PFM-based policy learns much more slowly. After 25 correct questions, the PFM-based policy's prediction changes by just over 0.1, and after 25 incorrect questions, the PFM-based policy's predictions changes by less than 0.03. In contrast, the BKT-based policy asymptotes over 10 questions to $1 - P(S) = 0.79$ when given correct responses and $P(G) = 0.47$ when given incorrect responses.

This figure also shows how the parameters of a BKT model affect decision making. $P(L_0)$ is responsible for the initial probability of a correct response. $P(S)$ and $P(G)$ respectively provide the upper and lower asymptotes for the probability of a correct response. $P(T)$ is responsible for the speed of reaching the asymptotes. For the PREDICTIVE SIMILARITY policy, the distance between the initial probability of a correct response and the asymptotes along with the speed of reaching the asymptotes is responsible for the number of questions suggested.

2.6 Discussion

Our results from experiment 1 show that the PREDICTIVE SIMILARITY policy performs similarly to the MASTERY THRESHOLD policy on BKT models with semantically meaningful parameters and suggests the same or fewer problems on BKT models without semantically meaningful parameters. Thus, this experiment suggests that the two instructional policies treat students successfully learning skills similarly. The lower ExpOps values for the PREDICTIVE SIMILARITY policy provide evidence that the PREDICTIVE SIMILARITY policy does not waste as much student time as the

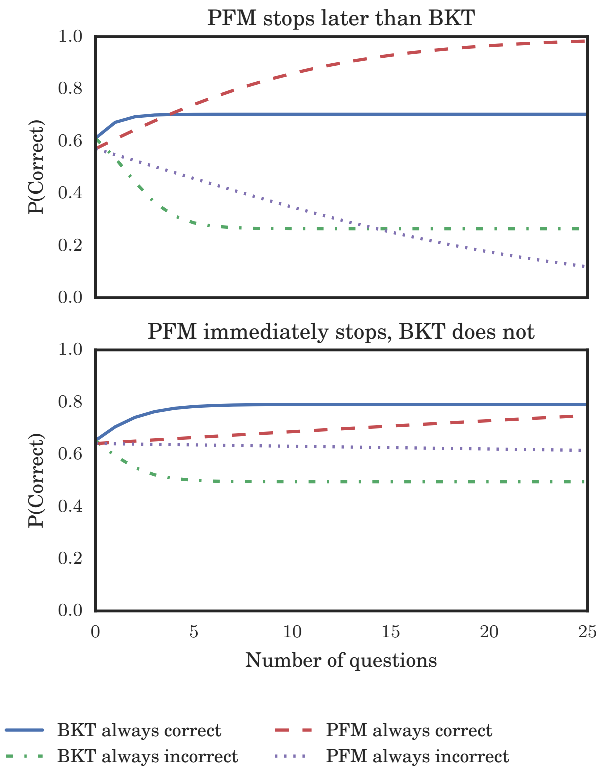


Figure 2.5: Predictions of BKT models and PFMs if given all correct responses or all incorrect responses on two skills.

MASTERY THRESHOLD policy on its own. Fundamentally, the MASTERY THRESHOLD policy fails to recognize that some students may not be ready to learn a skill. The PREDICTIVE SIMILARITY policy does not make the same error. Instead, the policy stops either when the system succeeds in teaching the student or when the skill is unteachable by the system. In practice MASTERY THRESHOLD policies are often used in conjunction with other rules such as a maximum amount of practice before stopping. A comparison of such hybrid policies to the PREDICTIVE SIMILARITY policy is an interesting direction for future work. However, it is important to note that such hybrid policies would still require the underlying model to have a notion of mastery, unlike our predictive similarity policy.

The PREDICTIVE SIMILARITY policy can be used to uncover differences in predictive models. Experiment 2 shows that policies based on models with the same predictive power can have widely different actions. AFMs had a very similar RMSE to both PFMs and the BKT models, but immediately stopped on a majority of the skills. An AFM must provide the same predictions to students who get many questions correct and students who get many questions incorrect. To account for this, its predictions do not change much over time. One may argue that this suggests that AFM models are poor predictive models, because their predictions hardly change with large differences in state. Both AFMs and PFMs have inaccurate asymptotes because it is likely that students who have mastered the skill will not get every question correct and that students who have not mastered the skill will not get every question incorrect. This means that these models will attempt to stay away from their asymptotes with lower learning rates. One possible solution would be to build LFMs that limit the history length. Such a model could learn asymptotes that are not 0 and 1.

2.7 Related Work

Predictive student models are a key area of interest in the intelligent tutoring systems and educational data mining community. One recent model incorporates both BKT and LFM into a single model with better predictive accuracy than both [13]. It assumes that there are many problems associated with a single skill, and each problem has an item parameter. If we were to use such a model in a when-to-stop policy context, the simplest approach would be to find the problem with the highest learning parameter for that skill, and repeatedly apply it. However, this reduces Khajah et al.'s model to a simple BKT model, which is why we did not explicitly compare to their approach.

Less work has been done on the effects of student models on policies. Fancsali et al. [10] showed that when using the MASTERY THRESHOLD policy with BKT one can view the mastery threshold as a parameter controlling the frequency of false negatives and false positives. This work focused on simulated data from BKT models. Since BKT assumes that students eventually learn, this work did not consider wheel-spinning. Rafferty et al. [21] showed that different models of student learning of a cognitive matching task lead to significantly different partially observable Markov decision process policies. Unlike our work which focuses on deciding when-to-stop teaching a single activity type, that work focused on how to sequence different types of activities and did not use a standard education domain (unlike our use of KDD cup). Mandel et al. [16] did a large comparison of different student models in terms of their predicted influence on the best instructional policy and expected performance of that policy in the context of an educational game; however, like Rafferty et al. their focus was on considering how to sequence different types of activities, and instead of learning outcomes they focused on enhancing engagement. Chi et al. [7] performed feature selection to create models of student learning designed to be part of policies that that would enhance learning gains on a physics tutor; however, the focus again was on selecting among different types of activities rather than a when-to-stop policy. Note that neither BKT nor LFMs in their original form can be used to select among different types of problems, though extensions to both can enable such functionality. An interesting direction of future work would be to see how to extend our policy to take into account different types of activities.

Work on when-to-stop policies is also quite limited. Lee and Brunskill [15] showed that individualizing student BKT models has a significant impact on the expected number of practice opportunities (as measured through ExpOps) for a significant fraction of students. Koedinger et al. [14] showed that splitting one skill into multiple skills could significantly improve learning performance; this process was done by human experts and leveraged BKT models for the policy design. Cen et al.[6] improved the efficiency of student learning by noticing that AFM models suggested that some skills were significantly over or under practiced. They created new BKT parameters for such skills and the result was a new tutor that helped students learn significantly faster. However, the authors did not directly use AFM to induce policies, but rather used an expert based approach to transform the models back to BKT models, which could be used with existing mastery approaches. In contrast, our approach can be directly used with AFM and other such models.

Our policy assumes that learning is a gradual process. If you were to instead subscribe to an all-at-once method of learning, you could possibly use the moment of learning as your stopping point. Baker et al. provide a method of detecting the moment at which learning occurs [2]. However, this work does not attempt to build instructional policies.

2.8 Conclusion & Future Work

The main contribution of this chapter is a when-to-stop policy with two attractive properties: it can be used with any predictive student model and it will provide finite practice both to students that succeed in learning a given skill and to those unable to do so given the presented activities.

This policy allowed us for the first time to compare common predictive models (LFMs and BKT models) in terms of their predicted practice required. In doing so we found that models with similar predictive error rates can lead to very different policies. This suggests that if they are to be used for instructional decision making, student models should not be judged by predictive error rates alone. One limitation of the current work is that only one dataset was used in the experiments. To confirm these results it would be useful to compare to other datasets.

One key issue raised by this work is how to evaluate instructional policy accuracy. One possible solution is to run trials with students stopping after different numbers of questions. The student would take both a pre and post-test, which could be compared to see if the student improved. However, such a trial would require many students and could be detrimental to their learning.

There is a lot of room for extending this instructional policy. First, we would like to incorporate other types of interactions, such as dictated information ("tells") or worked examples, into the PREDICTIVE SIMILARITY policy. This would give student models more information and hopefully lead to better predictions. Second, the PREDICTIVE SIMILARITY policy is myopic, and we are interested in the effects of expanding to longer horizons. In the next chapter, we will extend this work to choosing between skills. This will allow us to see if our model agnostic approach to policies will work on more complicated problems.

Chapter 3

Multiple Skills

3.1 Introduction

In the previous chapter, we gained two valuable insights. First, it is possible to build model agnostic instructional policies using a student model function interface. Second, latent factor models (LFMs) can be successfully used in instructional policies, but they suffer from their extreme asymptotes. In this chapter, we wish to extend our discussion of both of these claims by providing model agnostic instructional policies for the more complicated skill-choice problem and building new LFMs with flexible asymptotes.

The skill-choice problem can be defined as: given a set of skills and the amount of time left, which skill should the next question teach? If the skills being taught are independent of other skills, such as French vocabulary and fraction addition, then this problem can be easily solved using a when-to-stop policy. Simply teach the skill that will be mastered in the fewest steps given the when-to-stop policy. However in the general case, learning a skill may be dependent on having already mastered other skills. For example, we would not expect a student who has not mastered 1-digit addition to master 2-digit addition. This problem is complicated by the fact that we do not have access to which skills the student has mastered. We cannot just ask the student what they know. Instead, we have to estimate the student's mastery using their responses to questions. A solution to the skill-choice problem consists of two parts. First, it must capture the skill hierarchy (which skills each skill depends on) from student data. Then, it must provide a policy which decides which skill the next question should teach based on the number of questions left and the state of the student.

One solution is to use experts to both build the hierarchy and decide on the ordering of questions. Indeed this is often done in commercial systems, where sections are defined where all the skills in that section are independent and all of their prerequisites have been presented in sections the student must have mastered in order to reach or unlock this section. However, prior work has shown that in some cases experts may struggle with this task [16]. Instead, we split the problem so that a student model is responsible for capturing the skill hierarchy and an instructional policy uses the student model to decide which skill to teach next.

Our specific contributions in this chapter are as follows:

- We provide an updated function interface to student models that allows a student to learn more than one skill at a time (Section 3.4).
- We introduce two modifications to the performance factors models (Section 3.3). The first incorporates the skill hierarchy and the second allows for flexible asymptotes.
- We introduce two skill-choice policies using the updated function interface (Section 3.4). The first is a full horizon, computationally expensive algorithm that calculates the optimal skill to pick. The second is a greedy, computationally cheap algorithm that only consider short-term improvement.
- We describe a procedure to simulate student observations for a skill hierarchy using Bayesian knowledge tracing models (Section 3.5).
- We show that using flexible asymptotes substantially improves the quality of the parameters of the student model on long observation sequences (Section 3.6).
- We analyze the performance of the two policies using simulated data and the modified PFM

(Section 3.6). We find that the greedy policy tends to perform almost but not quite as well as the full horizon policy.

Our results agree with the work in the previous chapter that model agnostic policies are possible and useful, and that rigid predictive asymptotes limit the quality of student models. This work also shows that the extreme asymptotes of AFMs and PFMs are not necessary features of LFMs. The work described in this chapter is only preliminary. In future work, we would like to test our hypotheses on real data. We would also like to compare our new instructional policies against existing policies. Another interesting direction for future work would be to test the predictive performance of PFMs with flexible asymptotes.

3.2 A Short Aside: A Windowed PFM

Before we commence with the main topic of this chapter, we first briefly discuss the logistic regression asymptote issue. In the previous chapter, we saw that PFM predictions asymptote to 0 and 1, which negatively impacts instructional decision making. As the number of correct responses increases, the probability of a correct response asymptotes to 1, and as the number of incorrect responses decreases, the probability of an incorrect response asymptotes to 0. This does not match the expected behavior of students, because students who have not mastered the skill can accidentally guess the correct answer and students who have mastered the skill can make mistakes. For example, a mathematics professor accidentally writing the wrong symbol on the board does not mean that she has not mastered her subject, and a failing student responding correctly to a multiple choice question does not mean that she has mastered the subject of the test. In the previous chapter we conjectured that this led trained PFMs to have lower learning rates than they should. Thus, the extreme behavior of PFMs at their limits leads to poor instructional policies. In this section, we will propose a modification to PFM that prevents this behavior.

PFMs asymptote to 0 and 1 because the correct and incorrect counts for each skill are not constrained. Thus, one way improve asymptotic predictions is to constrain the counts. The Windowed

PFM (wPFM) only considers the past N responses for each skill. Instead of having parameters for the success and failure counts (μ_k and ρ_k), the windowed PFM has success and failure parameters for the last N responses (μ_k^N and ρ_k^N). Let s_k^N be the number of successful responses out of the previous N responses to questions for skill k, and let $f_k^N = N - s_k^N$ After the student has seen at least N questions, f_k^N equals the number of unsuccessful responses out of the previous N responses to questions for skill k. Since the model only considers the past N responses for each skill, predictions are bounded by the prediction given N incorrect responses and the prediction given N correct responses. The probability that student i will respond correctly to skill k with s_k^N correct responses in the past N questions for k is

$$P_{\text{wPFM}} = \frac{1}{1 - e^{-(\alpha_i + \mu_k^N s_k^N + \rho_k^N f_k^N)}} \tag{3.1}$$

Since wPFM is dimensionally smaller than the PFM, the skill parameter is unnecessary. It is subsumed by μ_k^N and ρ_k^N.

3.3 A Hierarchial PFM

In this chapter, we wish to use student models to capture the skill hierarchies. In the previous chapter, we saw that PFMs worked better with instructional policies than AFMs. Thus, we will use PFMs as the base model for policies in this chapter. However, the ideas described in this section could be generalized to all latent factor models. Unfortunately, PFMs do not share parameters between skills. This means that how well the student is doing on one skill has no impact on the model's prediction of success on other skills. Although not a problem in the previous domain, this makes the model useless for hierarchical skill modeling. Consider a student learning two skills, k_1 and k_2, where k_2 is dependent on k_1. Suppose student a responds correctly to 5 questions teaching k_1 and student b responds incorrectly to 5 questions teaching k_1. If both students have otherwise performed exactly the same, a PFM would say that they have the same probability of getting a question teaching k_2 correct, because predictions of performance on k_2 are not dependent on the student's performance on k_1. In this section, we modify PFM so that performance predictions for one skill are dependent on the student's performance on all skills.

3.3.1 Hierarchical PFM (hPFM)

The state of a PFM consists solely of the number of successful responses (s_k) and unsuccessful responses (f_k) the student has made to questions for each skill. When calculating the probability of a correct response for a given skill x, PFMs only consider s_x and f_x. The state of the other skills is not considered at all. The hierarchical PFM (hPFM) considers the state of all skills when calculating the probability of a correct response to a question for a given skill. Instead of only having parameters for the effects of the counts of the current skill, μ_k and ρ_k, it has $\mu_{k,l}$ and $\rho_{k,l}$ which correspond to the effect of the respective counts of skill l on skill k. The probability that student i will respond correctly to skill k given s_x correct responses and f_x incorrect responses for each skill x in the skill set K is

$$P_{\text{hPFM}}(C) = \frac{1}{1 - e^{-\left(\alpha_i + \beta_k + \sum_{l \in K}(\mu_{k,l}\, s_l + \rho_{k,l}\, f_l)\right)}} \tag{3.2}$$

Unlike PFMs, the number of parameters of hPFMs is quadratic in respect to the number of skills. Although this may seem like an substantial increase, the number of students tends to be considerably larger than the number of skills. Thus, the student parameters will tend have a larger role than the skill parameters in the size of the model.

3.3.2 Combining hPFM and wPFM

hPFM and wPFM are unrelated modifications of PFM. hPFM changes the structure of the parameters and wPFM changes the features for the parameters. This makes combining them quite straightforward. We will call the model that combines these modifications the hierarchical windowed PFM (hwPFM). The skill parameters for this model are $\mu^N_{k,l}$ and $\rho^N_{k,l}$. The probability that student i will respond correctly to skill k given s^N_x correct responses and f^N_x incorrect responses in the past N responses for each skill x in k is

$$P_{\text{hwPFM}}(C) = \frac{1}{1 - e^{-\left(\alpha_i + \sum_{l \in K}(\mu_{k,l}^N s_l^N + \rho_{k,l}^N f_l^N)\right)}} \tag{3.3}$$

In this paper, we are interested in working with hierarchical models. Therefore, we will only consider hPFM and hwPFM in the rest of this paper.

3.4 Model Agnostic Skill-Choice Algorithms

Student are frequently taught in a time-constrained environment. Student time is very valuable, and we wish to increase their learning as much as possible in the time alloted. To simplify the problem, we will assume that all questions take the same amount of time. Thus, the problem we wish to solve is: given a set of skills and time for T questions, what skill should the next question teach? In particular, we are interested in the case where there is a skill hierarchy, because learning independent skills is quite straightforward. A policy for this skill-choice problem should take in as input the state of the student and the maximum number of questions remaining. It should output either the next skill to teach or that the tutor should stop.

In this section, we provide two model agnostic skill-choice policies, one optimal and one greedy. But first, we must build up the foundational function interfaces.

3.4.1 An Updated Predictive Model Interface

In the previous chapter, we provided a function interface for predictive student models. This function interface was based on the premise of teaching one skill at a time and will not be powerful enough to capture learning multiple skills. Thus, we need to modify our function interface accordingly.

The single-skill predictive student model function interface consists of three functions: **startState**$(\dots)$, **updateState**$(\text{state}, \text{obs})$, and **predictCorrect**$(\text{state})$. For the multiple-skill function interface, we are going to keep the same functions, but add a skill argument to both **updateState** and **predictCorrect**. This changes their meaning slightly as described below:

startState$(\ldots)$**:** Provides the state of a student who has not seen any questions.

updateState$(\textbf{state}, \textbf{skill}, \textbf{obs})$**:** Updates the state given the correctness of the student response to a question on the given skill.

predictCorrect$(\textbf{state}, \textbf{skill})$**:** Predicts the probability of a correct response to a question for the given skill.

These function names tend to be a bit unwieldy in mathematical statements, so we will use some short hand.

$$up(\text{state}, \text{skill}, \text{obs}) := \textbf{updateState}(\text{state}, \text{skill}, \text{obs}) \tag{3.4}$$

$$P(C_{\text{skill}}|\text{state}) := \textbf{predictCorrect}(\text{state}, \text{skill}) \tag{3.5}$$

3.4.2 Score Interface

Tutor designers tend to have specific goals when building an intelligent tutoring system. The score function interface allows the goals of the tutor to be separated from the implementation of the skill-choice policy. A score provides a way of comparing the "goodness" of two states. This allows us to decide which state we would rather be in. For example, if someone is playing chess and their goal is to win, they would score a checkmate state higher than a check state. Instructional policies can use the score to tell how much they wish to be in a given state. They can then pick actions that maximize the score of either the next state or later states. The function interface for the score consists of a hidden score type, and three functions:

$sc(\textbf{state})$**:** The score of the given state.

$E(\textbf{score1}, \textbf{score2}, \textbf{p1})$**:** The expected score given that there is a p1 probability of having the score score1 and a $1 -$ p1 probability of having the score score2.

cmpScore$($**score1, score2**$)$**:** Returns whether score1 is better than, equal to, or worse than score2.

cmpScore$(\text{score1}, \text{score2})$ is inherently used in $\max$ and $\arg\max$ expressions.

Picking the correct scoring system is vital to the behavior of instructional policies. Later in this section, we will provide one possible scoring system. It is important to note that a scoring system may or may not be model agnostic. A scoring system may rely only on the model's function interface or rely on the inner workings of the model.

3.4.3 A Full Horizon Skill-Choice Policy

In this chapter, our goal is to pick the skill such that the expected score of the final state is the highest. Fortunately, the optimal policy according to this goal can be calculated. In artificial intelligence, planning algorithms make decisions based on their horizon. A planning horizon is how far ahead you look to measure the impacts of your decision. It is the number of times you consider all possible actions, consider all possible results of those actions, and repeat. One step of this procedure is shown in figure 3.1. To calculate the best move, one maximizes over each possible action and the calculates the expectation over each possible observation. This repeats until the planning horizon has been reached. A planning algorithm with a horizon of 1 would only look at the impacts of the decision on the possible next states. However, to calculate the optimal policy, one has to plan with a full horizon. This means that the impacts of the policy's decision at every possible step in the future must be taken into account. Unfortunately, this approach is computationally expensive, because it requires planning what to teach in every possible situation.

In order to mathematically define a full horizon policy, we first need to define a function that calculates the maximum expected score of the final state given a starting state and the number of questions left to teach. If there are no questions left to teach, then this function must return the score of the given state. If there are questions to teach, then this function must calculate the maximum expected final score recursively. In equation 3.6 we define this function with s being the state, t being the number of questions remaining, and K being the set of all skills.

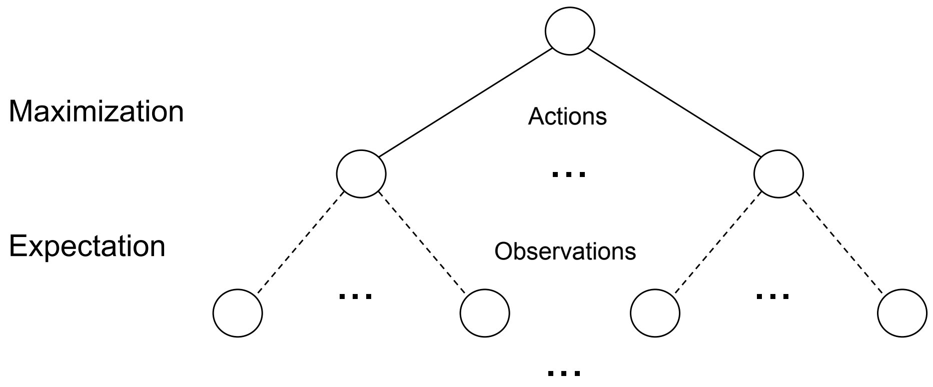


Figure 3.1: One step of the planning algorithm. This procedure is repeated recursively until the planning horizon is reached.

$$\textbf{expFinalScore}(\mathrm{s,t}) := \begin{cases} \begin{bmatrix} \max\limits_{k\in K} E(\textbf{expFinalScore}(up(s,k,C), t-1), \\ \qquad \textbf{expFinalScore}(up(s,k,\neg C), t-1), \\ \qquad P(C_k|s)) \end{bmatrix} & \text{if } t > 0 \\ sc(s) & \text{otherwise} \end{cases} \tag{3.6}$$

Now we can mathematically define the FULL HORIZON policy. The FULL HORIZON policy picks the skill that maximizes the expected final score. When there are no more steps left, the policy stops. This function can be defined as follows:

$$\textbf{fullHorizonPolicy}(\mathrm{s,t}) := \begin{cases} \begin{bmatrix} \arg\max\limits_{k\in K} E(\textbf{expFinalScore}(up(s,k,C), t-1), \\ \qquad \textbf{expFinalScore}(up(s,k,\neg C), t-1), \\ \qquad P(C_k|s)) \end{bmatrix} & \text{if } t > 0 \\ STOP & \text{otherwise} \end{cases} \tag{3.7}$$

The asymptotic complexity of this algorithm is $O((2|K|)^T)$ where T is the number of questions remaining. If there are many skills or the maximum number of questions is large, then this algorithm will be computationally prohibitive. We would like to find an algorithm that approximates the optimal policy and is computationally efficient.

3.4.4 A Greedy Skill-Choice Policy

The FULL HORIZON policy is computationally expensive, because it looks at every possible future situation of the student. Instead, let's consider a greedy approach that has a planning horizon of 1. The GREEDY policy picks the skill such that the expected score of the state after the question is the highest. This policy is not optimal, because it fails to take into account the later impact of the picked skill. However, we hope that the difference between using the GREEDY policy and the FULL HORIZON policy is small. We define the GREEDY policy as:

$$\textbf{greedyPolicy}(\mathrm{s},\mathrm{t}) := \begin{cases} \left[\begin{array}{l} \arg\max\limits_{k \in K} E(sc(up(s,k,C)), \\ \qquad\qquad sc(up(s,k,\neg C)), \\ \qquad\qquad P(C_k|s)) \end{array} \right] & \text{if } t > 0 \\ STOP & \text{otherwise} \end{cases} \tag{3.8}$$

Since the GREEDY policy only looks one step ahead, it has an asymptotic complexity of only $O(2|K|)$. This is far more reasonable than the exponential nature of the FULL HORIZON policy. We have traded optimality for reasonable computational complexity.

Although not discussed in this chapter, it is also possible to build skill-choice policies somewhere in between the GREEDY policy and the FULL HORIZON policy by looking n steps ahead. This policy should perform closer to optimal than the GREEDY approach at the expense of increasing the computational complexity to $O((2|K|)^n)$.

3.4.5 Generalizing Available Skills

As described above, both the FULL HORIZON and GREEDY policies assume that all policies should be allowed to pick any skill at any state. However, this assumption may not be true. For example, we may not wish to pick skills that we are confident the student has mastered. Fortunately, both the FULL HORIZON and GREEDY policies can be easily extended to allow for this generalization. First, we need a new function interface made of a single function **available**$(\text{state}, \text{skill})$ that returns whether or not the given skill is available to be shown next given the state. This function can be used at each step to filter out whether a skill is available or not.

The FULL HORIZON policy can be extended by filtering out unavailable skills at each step in both **expFinalScore** and **fullHorizonPolicy**. If there are no available skills, then the policy should stop. Let $K^s := \{k : k \in K,\ \textbf{available}(\text{s}, \text{k})\}$ be the set of skills available at state s. We can define the extended functions as:

$$\textbf{expFinalScore}'(\text{s}, \text{t}) := \begin{cases} \begin{bmatrix} \max\limits_{k \in K^s} E(\textbf{expFinalScore}'(up(s,k,C), t-1), \\ \qquad \textbf{expFinalScore}'(up(s,k,\neg C), t-1), \\ \qquad P(C_k|s)) \end{bmatrix} & \begin{array}{c} \text{if } t > 0 \text{ and} \\ |K^s| > 0 \end{array} \\ sc(s) & \text{otherwise} \end{cases} \tag{3.9}$$

$$\textbf{fullHorizonPolicy}'(\text{s}, \text{t}) := \begin{cases} \begin{bmatrix} \arg\max\limits_{k \in K^s} E(\textbf{expFinalScore}'(up(s,k,C), t-1), \\ \qquad \textbf{expFinalScore}'(up(s,k,\neg C), t-1), \\ \qquad P(C_k|s)) \end{bmatrix} & \begin{array}{c} \text{if } t > 0 \text{ and} \\ |K^s| > 0 \end{array} \\ STOP & \text{otherwise} \end{cases} \tag{3.10}$$

The GREEDY policy can be defined similarly by only considering available skills. As with the

FULL HORIZON policy, if there are no available skills, the GREEDY policy should stop. We can define the extended GREEDY policy as:

$$\mathbf{greedyPolicy'}(\mathrm{s}, \mathrm{t}) := \begin{cases} \begin{bmatrix} \arg\max_{k \in K^s} E(sc(up(s,k,C)), \\ \qquad sc(up(s,k,\neg C)), \\ \qquad P(C_k|s)) \end{bmatrix} & \text{if } t > 0 \text{ and } |K^s| > 0 \\ STOP & \text{otherwise} \end{cases} \tag{3.11}$$

The extended versions of the FULL HORIZON and GREEDY policies are a strict generalization of their unextended variants. We can get the original behavior by using an **available** implementation that always returns true. For the rest of this paper, we will only consider the extended variants of both policies.

3.4.6 A Score Implementation

There are many possible goals when teaching students. You may wish to maximize the number of skills mastered by the student given a budget of T steps, or you may wish to maximize the probability of a correct response from the student across all skills. For our score implementation, we considered the former. Under this goal, a state in which the student has mastered more skills is better than a state in which the student has mastered fewer. Suppose for now, that we have a mastery function interface including the function **mastered**(state, skill) that returns whether the given skill has been mastered in the given state. Later in this subsection, we will provide a model agnostic method of estimating student mastery. The mastery-count score will be the number of skills the student has mastered. This can be defined as

$$\mathbf{masteryCount}(\mathrm{state}) := \sum_{k \in K} \mathbb{1}(\mathbf{mastered}(\mathrm{state}, k)) \tag{3.12}$$

Many states that we think are different will have an equal score. Suppose a student is learning a single skill. All states before the student masters the skill will have the same mastery-count score. We were not content, because we believed that the GREEDY policy would suffer. The number of mastered skills does not change with every question. Thus, there could be situations where the score would be the same for every possible next state. The greedy policy would not be able to distinguish skills that the student could master from skills that the student could not.

Another possible score is be the average probability of a correct response over all the scores. This prediction-average score can be calculated as:

$$\textbf{predictionAvg}(\text{state}) := \frac{\sum_{k \in K} P(C_k | \text{state})}{|K|} \tag{3.13}$$

The prediction-average score has the benefit of separating states better than mastery-count score. Unfortunately, this score does not always match our goal. For example, consider the states shown in table 3.1. Suppose a student starts in state a and we get to decide if the student transitions to state b or state c. According to our goal, we would want to transition to state b, because state b has more mastered skills. However, according to the prediction-average score, we would transition to state c. Another problem with this score is that it counts improvements in predictions of mastered skills. Suppose two students mastered a skill. According to our goal, the states of the two students are equally valuable, regardless of the probability that the students respond correctly to the next question. However, the prediction-average score for the states of both students will depend on the probability that the students respond correctly to the next question.

Since our score function interface is general, we can easily merge the mastery-count score and the prediction-average scores. This combined score can be defined as the following tuple:

$$\textbf{combined}(\text{state}) := (\textbf{masteryCount}(\text{state}),\ \textbf{predictionAvg}(\text{state})) \tag{3.14}$$

The combined score is ordered first by the mastery-count score and then the prediction-average score. It does not suffer from lack of separation like the mastery-count score and it always scores

Table 3.1: Possible states to show differences in scores.

State a	k_1	k_2
$P(C)$	0.6	0.5
Mastered	No	No

Implementation	**Score**
masteryCount	0
predictionAvg	0.55
combined	$(0, 0.55)$
unmasteredPredictionAvg	0.55
limCombined	$(0, 0.55)$

State b	k_1	k_2
$P(C)$	0.8	0.5
Mastered	Yes	No

masteryCount	1
predictionAvg	0.65
combined	$(1, 0.65)$
unmasteredPredictionAvg	0.5
limCombined	$(1, 0.5)$

State c	k_1	k_2
$P(C)$	0.6	0.75
Mastered	No	No

masteryCount	0
predictionAvg	0.675
combined	$(0, 0.675)$
unmasteredPredictionAvg	0.675
limCombined	$(0, 0.675)$

states with more mastered skills higher, unlike the prediction-average score. Unfortunately, it still has a flaw. Consider the two states shown in table 3.2. Both states have a mastery-count of 1. State d has a higher prediction-average score thanks to a high prediction for the mastered skill. State e has a lower prediction-average than the first state, but has a higher prediction for the unmastered skill. Thus, state e is closer to mastering the unmastered skill than state d, but state d has a higher combined score. According to our goal, we would rather be in state e than state d.

Table 3.2: Possible states to show problems with combined score.

State d	k_1	k_2
$P(C)$	0.95	0.5
Mastered	Yes	No

Implementation	**Score**
masteryCount	1
predictionAvg	0.725
combined	$(0, 0.725)$
unmasteredPredictionAvg	0.5
limCombined	$(0, 0.5)$

State e	k_1	k_2
$P(C)$	0.8	0.6
Mastered	Yes	No

masteryCount	1
predictionAvg	0.7
combined	$(0, 0.7)$
unmasteredPredictionAvg	0.6
limCombined	$(0, 0.6)$

One possible solution to this problem is to remove the effect of mastered scores from the prediction-average score. This could be defined as the average prediction of skills that have not been mastered or the unmastered-prediction-average:

$$\textbf{unmasteredPredictionAvg}(\text{state}) := \frac{\sum_{k \in K} \mathbb{1}(\neg\textbf{mastered}(\text{state}, k)) P(C_k|\text{state})}{\sum_{k \in K} \mathbb{1}(\neg\textbf{mastered}(\text{state}, k))} \tag{3.15}$$

On its own this score is not particularly useful. However, just like the prediction-average score, it works well in conjunction with the mastery-count score. We define the tuple of the mastery-count score and the unmastered-prediction-average score as the limited-combined score:

$$\textbf{limCombined}(\text{state}) := (\textbf{masteryCount}(\text{state}),\ \textbf{unmasteredPredictionAvg}(\text{state})) \tag{3.16}$$

The limited-combined score will view improvements in the probability that a student will respond correctly to a mastered skill as inconsequential. It will score states with more mastered skills as higher than states with fewer mastered skills. It will also score a state higher than another state with the same number of mastered states if it has a higher probability of a correct response on an unmastered skill, because that suggests the this state is closer to mastering that skill. This effect can be seen in table 3.2. In this chapter, we will use the limited-combined score with our policies. The implementation of the function interface can be found in equations:

$$sc(s) := \textbf{limCombined}(s) \tag{3.17}$$

$$E((m_1, n_1), (m_2, n_2), p_1) := (p_1 m_1 + (1 - p_1) m_2, p_1 n_1 + (1 - p_1) n_2) \tag{3.18}$$

$$\textbf{cmpScore}((m_1, n_1), (m_2, n_2)) := \begin{cases} \text{Greater} & \text{if } m_1 > m_2 \\ \text{Greater} & \text{if } m_1 = m_2 \text{ and } n_1 > n_2 \\ \text{Equal} & \text{if } m_1 = m_2 \text{ and } n_1 = n_2 \\ \text{Less} & \text{otherwise} \end{cases} \tag{3.19}$$

We must now consider the issue of calculating student mastery of a skill. Some student models, such as BKT, maintain a probability of mastery ($P(L_k)$) for each skill. As shown in the previous chapter, the probability of mastery can then be used along with a confidence threshold to provide an estimate of mastery. For state s and skill k, the confidence threshold mastery function is:

$$\textbf{confidenceThreshold}(s, k) := P(L_k|s) \geq \Delta \tag{3.20}$$

Unfortunately many models, such as PFM, only provide the probability of a correct response for the next question. One solution is to simply use a threshold on the probability of a correct response. For a BKT model, the confidence threshold mastery function can be written in terms of the probability of a correct response rather than the probability of mastery.

$$\textbf{confidenceThreshold}_{BKT}(s, k) \equiv P(C_k|s) \geq (1 - P(S_k) - P(G_k))\Delta + P(G_k) \tag{3.21}$$

This can be generalized to a prediction threshold mastery function:

$$\textbf{predictionThreshold}(s, k) := P(C_k|s) \geq \delta \tag{3.22}$$

The prediction threshold mastery function has the benefit of being model agnostic. It only requires the predictive student model function interface. Unfortunately, it may not always be a good estimator of mastery. The best threshold may not be the same for all skills. Suppose you are teaching one skill with multiple-choice and another with fill-in-the-blank. If a student tends to answer fill-in-the-blank questions correctly, we can be quite confident that the student has mastered the skill. However, a student who tends to answer multiple-choice questions correctly k could just be getting lucky, because multiple-choice questions are much easier to guess. In this chapter, we use the prediction threshold mastery function, but we leave the door open for better model-agnostic implementations and model-dependent implementations.

Finally, we must provide an implementation of **available**$(\text{state}, \text{skill})$. Since our goal is to master as many skills as possible, we do not want to waste time teaching skills that have already been mastered. We assume that once a skill has been mastered, practicing that skill has no impact on student performance on other skills. Thus, we implement **available** given state s and skill k as:

$$\textbf{available}(s, k) := \neg\textbf{mastered}(s, k) \tag{3.23}$$

This forces the algorithm to only teach skills that the student has not mastered.

3.5 Simulating Students

Testing skill-choice policies with real student data is complex, expensive, and time consuming. For our preliminary results, we elected to instead use simulated student data. Simulating student data involves stochastically generating observation sequences. For our experiments, we randomly selected which skill the student would learn at each step. These observation sequences could then be used to train student models. In this section, we describe a simulation implementation using BKT models. We decided to use BKT models because they can be easily used in simulations, they maintain a state of mastery, and because we were testing LFMs, which have a very different structure.

3.5.1 Single Skill BKT Simulation

Let's first consider the simpler problem of simulating a student learning a single skill. According to the assumptions made by BKT, a student can be in one of two states: mastery or non-mastery. The student begins in the mastery state with a probability of $P(L_0)$. At each question, if the student has not mastered the skill, the student has a $P(T)$ probability of mastering the skill. If the student has not mastered the skill, they will respond correctly with a probability of $P(G)$. If the student has mastered the skill, they will respond correctly with a probability of $1 - P(S)$. Given $P(L_0)$, $P(T)$, $P(G)$, and $P(S)$, one can simulate a student's behavior.

The simulation algorithm only has to track whether or not the student has mastered the skill. The initial state of mastery is stochastically computed such that there is a $P(L_0)$ probability that the simulated student has mastered the skill. At each step, if the student has not reached mastery, the algorithm stochastically transitions to the mastery state with a probability of $P(T)$. If, after the possibility of transition, the student is in the mastered state, the algorithm stochastically states that the student responds correctly to the question with a probability of $1 - P(S)$. Otherwise, it states that the student responds incorrectly. If the student is in the unmastered state, the algorithm stochastically states that the student responds correctly to the question with a probability of $P(G)$. This algorithm allows us to simulate a student learning a skill over any time period. We simply input the BKT model parameters and the length of the observation sequence, and the algorithm will output a stochastic observation sequence.

3.5.2 Hierarchical Skill BKT Simulation

Now that we have a single skill BKT simulation algorithm, we can build a multiple skill BKT simulation algorithm. We wish to simulate a student learning multiple skills at the same time. In this chapter, we will stochastically pick which skill to teach at each step. The BKT multiple skills simulation algorithm takes as input BKT parameters for each skill and the number of questions to be given to the student. It maintains the mastery state of each skill and initializes the state of each student in the same way as the single skill BKT simulation. At each step, a skill is stochastically chosen. Then, the step is simulated in the same way as the single skill BKT simulation with the BKT parameters for the chosen skill. The outputs of the algorithm are the chosen skill and the student observation for each question.

The multiple skill BKT simulation algorithm works well in a situation where all skills are independent. For example, it could simulate a student learning a skill in both history and mathematics at the same time. However, it fails to capture skill hierarchy. The simulated ability of the student to master the skill being currently taught is in no way dependent on whether any other skill has been mastered. The hierarchical skill BKT simulation algorithm modifies the multiple skill BKT simulation to take a skill hierarchy into account. Along with the BKT parameters for each skill and the number of questions, the hierarchical skill BKT simulation algorithm takes a skill hierarchy as input. The skill hierarchy allows the algorithm to tell which skills a specific skill is dependent on. The hierarchical skill BKT simulation algorithm is the same as the multiple skill BKT simulation algorithm except that a student can only master a skill if all skills it is dependent on have been mastered. This prevents a student from mastering a skill before its parents have been mastered. All simulated observation sequences in this chapter will be computed using the hierarchical skill BKT simulation algorithm.

3.6 Experiments

We now provide two preliminary experiments based on simulated data. The first compares the parameters of trained hPFMs and hwPFMs to see the effect of windowing. The second compares the FULL HORIZON and GREEDY skill-choice policies to see the effects of being myopic.

3.6.1 Model Implementation

The hPFMs and hwPFMs were implemented using sci-kit learn's logistic regression [20]. We used L1 normalization and included a fit intercept. The tolerance was 10^{-4}. The simulated data was computed using a python package we developed. Both the FULL HORIZON and GREEDY policies were implemented as python functions.

3.6.2 Experiment 1: hPFM and hwPFM Parameter Comparison

In the previous chapter, we saw that the PFM struggled due to the asymptotic nature of its predictions. As the student answered more questions correctly, the model's prediction of the student responding correctly to the next student approached 1. These predictions do not match our intuitions of student behavior, because students slip up on subjects they are very knowledgeable on for a variety of factors. This structural problem provides predictions as though students will continuously improve. A student who masters a skill after 5 steps will be predicted to have a much higher probability of responding correctly after 200 steps even though this does not match our beliefs about student behavior. We hypothesize that this leads PFM parameters to be dependent on the length of the observation sequences that they were trained on, because if the model has to provide reasonable predictions at 200 steps, then they will learn a much lower learning rate than if they only need to provide reasonable prediction for up to 10 steps. We do not want student models to be dependent on the lengths of the observation sequences, because they are only supposed to capture student behavior on questions. Observation sequence length is not a feature of student behavior. In this experiment, we wish to see if hPFMs are dependent on the lengths of the observation sequences they were trained on and whether windowing has an impact on this effect.

Parameters of latent factor models such as hPFMs and hwPFMs can be separated into two groups: stationary parameters that only have feature values of 0 and 1 and variable parameters that can have many different feature values. Stationary parameters are properties of the student and skills that do not change over time. Student aptitude and skill difficulty are both stationary parameters. Variable parameters are properties that change over time. For example, the counting parameters such as $\mu_{k,l}$ and $\mu^N_{k,l}$ are variable parameters.

To capture the effect of the length of training observation sequences on the parameters of the models, we trained both hPFMs and hwPFMs on sets of simulated observation sequences of different lengths. The skill taught at each step was chosen randomly. This allowed us to see how parameters changed with respect to the lengths of training observation sequences. Specifically, for each observation sequence length, we simulated 10 sets of 1000 observation sequences and trained both a hPFM and a hwPFM on each set of observation sequences. Both skills were simulated as having the same BKT parameters, namely $(P(L_0) = 0, P(T) = 0.2, P(G) = 0.25, P(S) = 0.1)$. We used a $P(L_0) = 0$ so that the data would be simulated as though the student had not mastered either skill. We then took the mean of the parameters for each observation sequence length and plotted them as shown in figure 3.2.

The hPFM plots show that all the variable parameters approach 0 as the length of the training

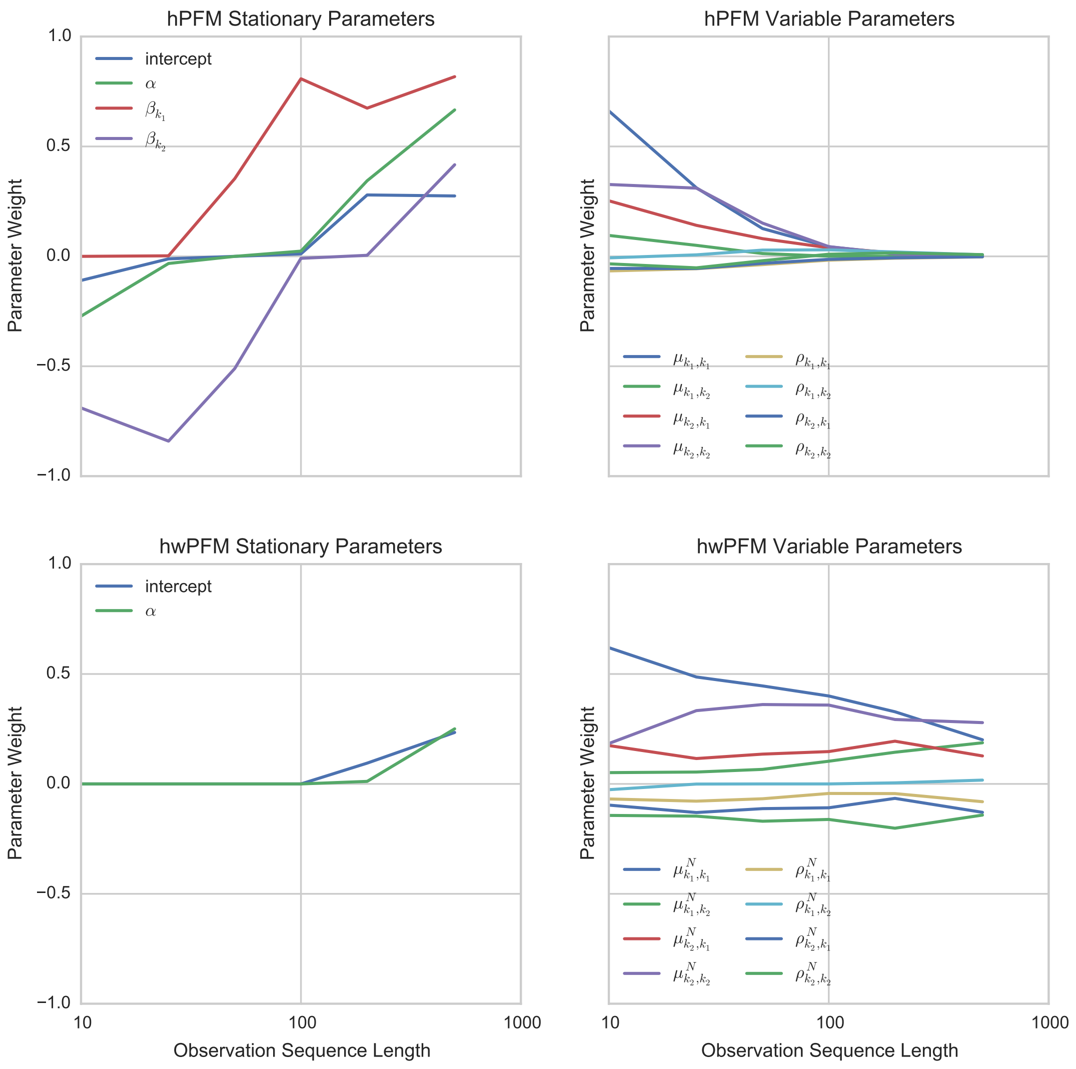


Figure 3.2: Comparison of both the stationary and variable parameters of hPFMs and hwPFMs trained on observation sequences of different lengths

observation sequences increases. This means that student responses become less influential in prediction as the length of the training observation sequences increases. The stationary parameters become the only influential parameters. An hPFM trained on long observation sequences only has two important features: the skill and the student. Fortunately, the hwPFM does not suffer from the same fate. Certain parameters such as the effect of the windowed count of correct responses to k_1 on predicting the probability of a correct response to k_1 ($\mu^N_{k_1,k_1}$) and the effect of the windowed count of correct responses to k_2 on predicting the probability of a correct response to k_1 ($\mu^N_{k_1,k_2}$) do change, but their values are both semantically meaningful with short and long training observation sequences. The stationary parameters of the hwPFM are 0 when the length of training observation sequences is less than 100 and increases afterwards, although not as much as the hPFM stationary parameters. This suggests that with long training observation sequences, predictions have a bias towards correctness. One possible reason for this is that the student has mastered the skill for most of the observation sequence leading to more correct responses being in the observation sequence.

3.6.3 Experiment 2: Comparing Greedy and Full Horizon Policies

The GREEDY policy gives up proven optimality for speed. We wished to see how much, if any, instructional performance the GREEDY policy gives up in this deal. To do so, we looked at simple skill hierarchies and compared how the GREEDY and FULL HORIZON policies fared at teaching each skill in the hierarchy. Since students learn stochastically, we compared the expected number of mastered skills after following the two policies. We calculated these values for different numbers of steps so that we could see how mastery improves with more steps available to the policy.

Before describing our methodology, we must first define decision graphs. A policy's actions can be described as a graph where the nodes are the questions given to the student and the branches are the possible student observations (correct and incorrect). At each node, the skill chosen by the policy is placed. The first node is the first question that would be given to the student. From here, the graph branches out on the two possible student observations. This leads to two nodes, one given a correct response and one given an incorrect response. At each node, the skill chosen by the policy is placed and from this point the graph is recursively filled in until the maximum number of problems is reached. A decision graph along with a student model can be used to calculate the expectation of mastery of a skill. One calculates the probability of each leaf node by traversing the graph and calculating the probability of each observation. Then, one can use the student model to see if the student has mastered the skill at the leaf. We used the prediction threshold implementation of mastery described in Subsection 3.4.2 with $\delta = 0.75$. The number of mastered skills can then be calculated by summing the expectation of mastery for each skill.

We performed our experiment on multiple skill hierarchies to see how the hierarchy affected both the GREEDY and FULL HORIZON policies. In particular, the policies we tested were:

single: A single skill.

double: Two skills where one ‘child’ skill is dependent on the other ‘parent’ skill.

triple: Three skills where one ‘grandchild’ skill is dependent on a ‘child’ skill which is dependent on a ‘parent’ skill.

separate: Three skills where all skills are not dependent on any skill.

fork: Three skills where two ‘child’ skills are dependent on a ‘parent’ skill.

together: Three skills where a ‘child’ skill is dependent on two ‘parent’ skills.

To generate our results, we ran the same experiment on each skill hierarchy. We first produced 1000 simulated observation sequences of length 50 using the skills in the skill hierarchy. Every skill was modeled with the same BKT model parameters: $(P(L_0) = 0, P(T) = 0.4, P(G) = 0.25, P(S) = 0.1)$. We used a higher transition probability than in the previous experiment so that we could simulate a student mastering more skills with less questions. We then trained an hwPFM on the simulated observation sequences. For a variety of maximum numbers of questions, we built decision trees using the FULL HORIZON policy and the GREEDY policy. We then used these decision trees to calculate the expectation of mastery for each skill in the hierarchy. These values are plotted in figures 3.3 and 3.4.

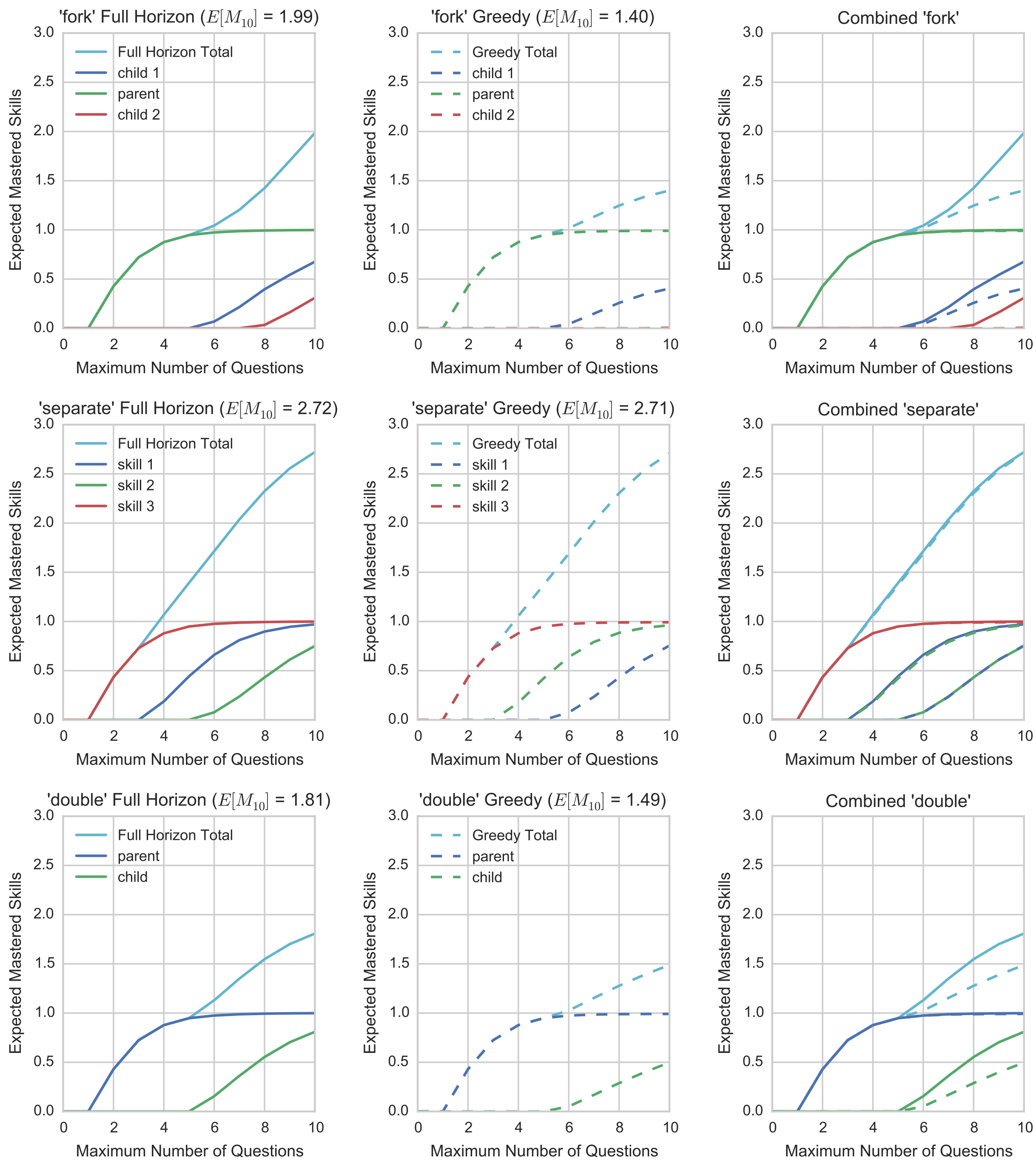


Figure 3.3: The expected mastery for each skill hierarchy given different maximum numbers of questions. ‘fork’ contains two children skills who are dependent on a parent skill. ‘separate’ contains three independent skills. ‘double’ contains a child skill which is dependent on a parent skill. The combined plots are the greedy plots superimposed on the full horizon plots.

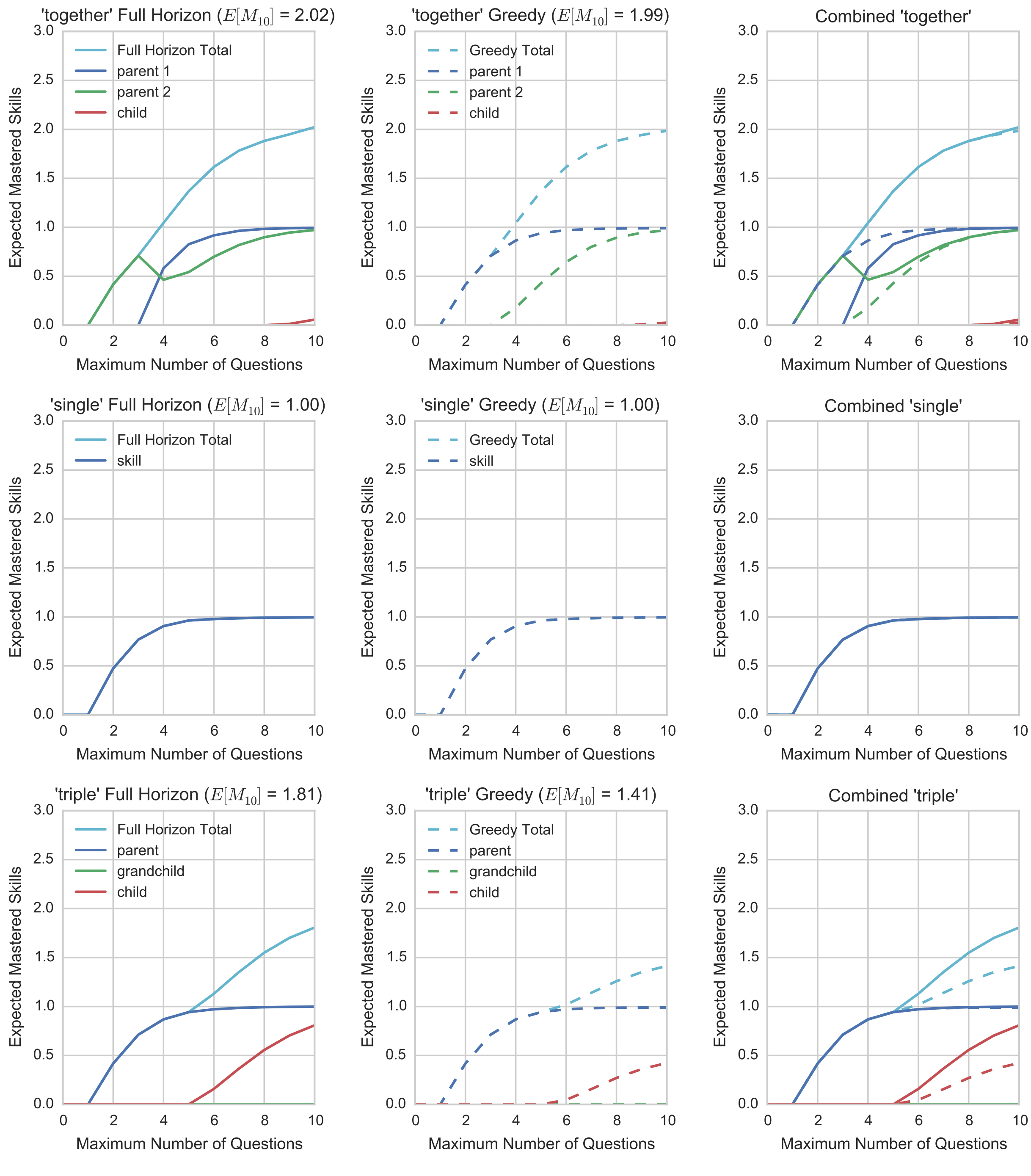


Figure 3.4: The expected mastery for each skill hierarchy given different maximum numbers of questions. 'together' contains a child skill which is dependent on two parent skills. 'single' contains a single skill. 'triple' contains a grandchild skill which is dependent on a child skill which is dependent on a parent skill. The combined plots are the greedy plots superimposed on the full horizon plots.

These plots show that the GREEDY policy tends to agree with the optimal policy. The GREEDY policy is almost optimal when the maximum number of questions is small. However, as the maximum number of questions increases, the FULL HORIZON policy begins to pull away from it. These plots also show that the expectation for each skill increases monotonically for the GREEDY policy but for not the FULL HORIZON policy. In fact, the FULL HORIZON policy sometimes seems to sacrifice the expected mastery of one skill for a later skill. In the 'together' skill hierarchy the FULL HORIZON policy sacrifices the expected mastery of a parent skill for the expected mastery of another parent. The GREEDY policy is incapable of making such sacrifices. However, this does not seem to have a large impact on student performance, because the two policies have a similar expected number of mastered skills on the 'together' skill hierarchy.

3.7 Discussion

Our results from experiment 1 suggest that windowed LFMs do not suffer from the asymptotic issues that regular LFMs do. The variable parameters of the hPFM went to 0, because that was the only way that the model could asymptote to predictions other than 0 and 1. As the length of the training observation sequences increased, the hPFM was downgraded to a predictive model that simply said that students are more likely to respond correctly to questions teaching k_1 than questions teaching k_2. Fortunately, the hwPFM did not suffer such an embarrassing fate, because its predictions can asymptote to any value.

Certain parameters of the hwPFM did change as the length of the training observation sequences increased. The most obvious change was the decrease of $\mu^N_{k_1,k_1}$. This parameter is responsible for the effect of getting previous questions teaching k_1 correct on predicting the probability of a correct response if teaching k_1. One would expect this parameter to be high, because answering correctly to previous questions on a skill is a good indicator that the student has mastered the skill. This change could be due to increases in other parameters, such as the two static parameters. The $\mu^N_{k_1,k_2}$ parameter increased as the length of the training observation sequences did. This parameter is responsible for the effect of getting previous questions teaching k_2 correct on predicting the probability of a correct response if teaching k_1. k_2 is dependent on k_1, which means that if the student has mastered k_2 the student must have also mastered k_1. Since responding correctly to questions teaching k_2 is a good indicator that the student has mastered k_2, it is also a good indicator that the student has mastered k_1. With short training observation sequences, all k_2 questions may get in the way of mastery of k_1. With longer training observation sequences, more correct k_2 questions may become an indicator that k_1 is mastered leading to an increase in $\mu^N_{k_1,k_2}$. Another interesting parameter is $\rho^N_{k_1,k_2}$, which is responsible for the effect of getting previous questions teaching k_2 incorrect on predicting the probability of a correct response if teaching k_1. When trained on short observation sequences, $\rho^N_{k_1,k_2}$ is negative, but when trained on long observation sequences, it is positive. In short observation sequences, there is very little time for k_1 to be mastered. Thus, any question teaching k_2 could have a negative impact, because the k_2 question is taking up a question that could have been used by k_1. In long observation sequences, the effect of questions teaching k_2 taking up space is dwarfed by the number of questions taught where the student has already mastered k_1. Unfortunately, it is hard to say which trained parameters are the best. They were all taught on data simulated from the same student models that they were meant to be capturing. Fortunately, the parameter changes are small, at least in comparison to the hPFM.

The seemingly better results gained from using a hwPFM over a hPFM require an extra tuned parameter: the window length. Unfortunately, we have not yet found a reasonable way of deciding the best window length.

Our results from experiment 2 suggest that the GREEDY policy performs almost optimally on small numbers of questions. When only a small number of questions are available, the FULL HORIZON policy can only look ahead a few questions further than the GREEDY policy making the benefits of the FULL HORIZON policy quite small. As the number of questions increases, the difference in the amount of lookahead grows which causes the GREEDY policy to fall behind the optimal policy. As the number of questions increases, the expectation of mastery for each skill must monotonically increase. The optimal policy is not bound by this constraint, which means that it can move resources from one skill to another if that will increase the expected number of mastered skills.

Our results also show patterns in the policies created when using a hwPFM with our score implementation. The simplest skill hierarchy, ‘single’, suggests that when learning a single skill, the probability of mastery first grows quickly and then plateaus. Although unsurprising, this shows why it is important to either teach multiple skills at once or have a good when-to-stop policy. Otherwise, students may practice skills for too long, which wastes their time. The results for the ‘separate’ skill hierarchy also back this claim. Learning unrelated skills led to a continuous linear increase in the expected number of mastered skills. If the student is allowed to learn multiple skills in a session, then after mastering a skill, they can begin learning a new skill. This prevents the student from waisting their time. The GREEDY policy acted optimally for the ‘separate’ skill hierarchy, which suggests that dependences are what trip up the GREEDY policy. One interesting feature is that the ‘double’ skill hierarchy suggests that dependent skills are not learned as quickly as independent skills. Unlike the ‘separate’ skill hierarchy, the expected number of mastered skills slows as the gains from the expectation of mastery of the parent skill die off. The child skill begins to have a chance of being mastered when the FULL HORIZON policy is given 6 questions. The ‘separate’ skill hierarchy only requires 4 questions before the second skill shows a chance of being mastered. This could be due to the fact that the policy has to be more confident that the parent skill is mastered, because if it moves on too quickly, then not only is the parent not mastered, but neither are any of the parent’s descendants. The ‘double’ skill hierarchy is also the first hierarchy in which the GREEDY policy performs noticeably worse than the FULL HORIZON policy. Although expectation of mastering the parent skill stays the same, the expectation of mastering the child skill is noticeably lower. This bolsters the claim that the FULL HORIZON policy struggles when handling dependent skills. We also found that the ‘double’ and ‘triple’ skill hierarchies had similar behavior. This is reassuring, because the policy does not have enough steps before it can master the grandchild skill in the ‘triple’ skill hierarchies. Both the FULL HORIZON and GREEDY policies have the same behavior on these two skill hierarchies. One of the most interesting results from this experiment was the characteristics of the FULL HORIZON policy on the 'together' skill hierarchy. On the fifth step, some of the expected mastery of a parent skill is sacrificed for the expected mastery of another parent skill. By looking ahead, the algorithm is able to master the second skill faster. Interestingly, this has very little impact on the expected number of mastered skills. However, it does show that the FULL HORIZON policy is capable of making decisions that the GREEDY policy cannot.

3.8 Related Work

Learning skill hierarchies from data has been a key area of interest for decades. Gagne was the first to describe the prerequisite structure of skills as learning hierarchies [11]. He also developed methods for learning hierarchies from data. More recently, Brunskill provided a method of estimating a skill hierarchy from noisy data [4]. This work involved using student observations to estimate the skill hierarchy. The skill hierarchy could then be used as part of an intelligent tutoring system to drive decision making. Unlike our work, Brunskill's work was focused on outputting a human-readable skill hierarchy. Our work is instead interested in building a student model that can capture the effects of a skill hierarchy on student performance. There does not have to be an easy way of outputting a human-readable skill hierarchy.

The work in this thesis is based on the premise that building model agnostic instructional policies will allow us to use powerful predictive student models in decision making. Another approach would be to build better models for the instructional policies we already have. González-Brenes et al. suggested an extended knowledge tracing model that integrates general features into Bayesian Knowledge Tracing. This extended model improved prediction performance considerably.

3.9 Conclusion

The two main contributions of this chapter are that our modular approach to policy building works on a harder problem than the when-to-stop problem and that comparing the performance of student models in decision making provides valuable insights into both the policy and the model. We implemented two skill-choice policies using function interfaces that made the policies both model agnostic and goal agnostic. This allowed us to split the problem into pieces. We designed the model, score, and skill chooser separately. This means that a tutoring system designer can pick and choose from our work. They can use different goals and better student models if they wish. They can also change their assumptions about which skills should be taught at a given state. The dependence of the three components is also left up to the designer. This means that simple predictive models can be used, but so can more complicated models with larger function interfaces. These function interfaces can then be used to implement the score and skill chooser.

The behavior of latent factor models in the previous chapter pushed us to design new models with novel characteristics. We built the wPFM that does not suffer from the extreme predictive asymptotes we saw in the previous chapter. Attempting to design a policy for the skill-choice problem also pushed us to design the hPFM, because the PFM will not capture skill hierarchies. If we had only been concerned with model accuracy, we may never have considered these modifications.

More generally, this work shows that it is possible to approach artificial intelligence problems, specifically in the intelligent tutoring domain, in a modular way. This may lead both to better models and more flexible policies.

Chapter 4

Conclusion

4.1 Future Work

The results and ideas expressed in Chapter 3 are only preliminary and should be taken lightly. Although simulated data is useful, it is not a substitute for real student data. Simulated data includes all the assumptions made by the underlying model. This can make signals appear cleaner than they do in real data. Thus, we would like to run the two experiments described in Chapter 3 on real student data. For the second experiment, this may require collecting new data as using previously collected data to compare policies can be very difficult.

PFMs are frequently used for predicting student performance. However, we did not judge our new models on their predictive performance. Thus, we would like to compare all four models on a large dataset, such as the KDD Cup dataset used in Chapter 2. If the wPFM performs well, we would like to see how it fares with the PREDICTIVE SIMILARITY policy. As mentioned previously, a large drawback of the windowed PFM is that it requires another tunable parameter. One interesting avenue for future work would be to see if this parameter can be either trained or chosen methodically. Experiment 1 showed that hierarchical PFMs are able to capture information about skill hierarchy. It may be possible to use the parameters or predictions from trained hierarchical PFMs to learn the skill hierarchy from data.

Judging and comparing policies is notoriously difficult. Offline approaches tend to either require large amounts of data or lead to statistically inconsequential results. Thus, one avenue for future work would be to test these policies by teaching students according to both policies and calculating how effective they are. One would have to not only compare the policies, but also see how the maximum number of questions affects student performance. The FULL HORIZON policy and the GREEDY policy are both extreme. The FULL HORIZON policy is optimal but prohibitively expensive, and the GREEDY policy is fast but myopic. It would be interesting to see how policies in between these two perform and at what rate increasing computational cost increases performance. We considered a simplified problem where each question has the same cost. In future work, we would like to expand our policies to handle a situation where questions for different skills have different costs. The policies could also be expanded to handle other types of interactions, such as "tells". Finally, we would like to compare the effects of different score implementations on the policies. In these experiments, we only tested one score implementation. However, this could be the most important aspect of the policy, so understanding its effects on policies could be tremendously helpful.

4.2 Conclusion

In this thesis, we provided the following contributions. First, we proposed a novel model agnostic when-to-stop policy that prevents wheel-spinning. Our results suggest that this policy performs similarly to the industry standard mastery-threshold policy, except that it is able to stop sooner on students who cannot learn with the given instruction. Second, we provided evidence that the extreme prediction asymptotes of PFM harm instructional decision making with the model. This also showed that instructional policies based on models with similar predictive performance can make wildly different decisions. If student models are to be used in instructional policies, predictive accuracy is not a strong measure on its own. Third, we proposed a windowed PFM that does not suffer from the extreme prediction asymptotes of PFM. Fourth, we proposed a hierarchical PFM that, unlike a normal PFM, is able to capture a skill hierarchy. Finally, we proposed two new model agnostic skill-choice algorithms that make decisions based on a generic score. This allows tutor designers to use whatever predictive models and goals they wish without having to design a new instructional policy.

The main conclusion of this thesis, however, is that it is entirely possible to build model agnostic instructional policies and that doing so will not only give you the freedom to use any predictive student model, but also provide a powerful way to compare student models.

Bibliography

[1] V. Aleven, B. M. McLaren, J. Sewall, and K. R. Koedinger. The cognitive tutor authoring tools (ctat): preliminary evaluation of efficiency gains. In *Intelligent Tutoring Systems*, pages 61–70. Springer, 2006.

[2] R. S. Baker, A. B. Goldstein, and N. T. Heffernan. Detecting learning moment-by-moment. *IJAIED*, 21(1):5–25, 2011.

[3] B. S. Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational researcher*, pages 4–16, 1984.

[4] E. Brunskill. Estimating prerequisite structure from noisy data. In *EDM*, pages 217–222, 2011.

[5] H. Cen, K. Koedinger, and B. Junker. Learning factors analysis–a general method for cognitive model evaluation and improvement. In *Intelligent tutoring systems*, pages 164–175. Springer, 2006.

[6] H. Cen, K. R. Koedinger, and B. Junker. Is over practice necessary?-improving learning efficiency with the cognitive tutor through educational data mining. *FAIA*, 158:511, 2007.

[7] M. Chi, K. R. Koedinger, G. J. Gordon, P. W. Jordan, and K. VanLehn. Instructional factors analysis: A cognitive model for multiple instructional interventions. In *EDM*, pages 61–70, 2011.

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

[9] M. H. Falakmasir, Z. A. Pardos, G. J. Gordon, and P. Brusilovsky. A spectral learning approach to knowledge tracing. In *EDM 2013*, pages 28–35, 2010.

[10] S. E. Fancsali, T. Nixon, and S. Ritter. Optimal and worst-case performance of mastery learning assessment with bayesian knowledge tracing. In *Proceedings of the 6th International Conference on Educational Data Mining*, 2013.

[11] R. M. Gagne, W. W. Wager, K. C. Golas, J. M. Keller, and J. D. Russell. Principles of instructional design, 2005.

[12] Y. Gong, J. E. Beck, and N. T. Heffernan. Comparing knowledge tracing and performance factor analysis by using multiple model fitting procedures. In *Intelligent Tutoring Systems*, pages 35–44. Springer, 2010.

[13] M. Khajah, R. M. Wing, R. V. Lindsey, and M. C. Mozer. Integrating latent-factor and knowledge-tracing models to predict individual differences in learning. *EDM*, 2014.

[14] K. R. Koedinger, J. C. Stamper, E. A. McLaughlin, and T. Nixon. Using data-driven discovery of better student models to improve student learning. In *AIED*, pages 421–430. Springer, 2013.

[15] J. I. Lee and E. Brunskill. The impact on individualizing student models on necessary practice opportunities. In *EDM*, 2012.

[16] T. Mandel, Y.-E. Liu, S. Levine, E. Brunskill, and Z. Popovic. Offline policy evaluation across representations with applications to educational games. pages 1077–1084. AAMAS, 2014.

[17] K. Milligan, E. Moretti, and P. Oreopoulos. Does education improve citizenship? evidence from the united states and the united kingdom. *Journal of Public Economics*, 88(9):1667–1695, 2004.

[18] Z. A. Pardos and N. T. Heffernan. Modeling individualization in a bayesian networks implementation of knowledge tracing. In *User Modeling, Adaptation, and Personalization*, pages 255–266. Springer, 2010.

[19] P. I. Pavlik Jr, H. Cen, and K. R. Koedinger. Performance factors analysis–a new alternative to knowledge tracing. 2009.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

[21] A. Rafferty, E. Brunskill, T. Griffths, and P. Shafto. Faster teaching by POMDP planning. In *AIED*, 2011.

[22] J. Stamper, A. Niculescu-Mizil, S. Ritter, G. Gordon, and K. Koedinger. Algebra i 2008-2009. challenge data set from kdd cup 2010 educational data mining challenge. find it at http://pslcdatashop.web.cmu.edu/kddcup/downloads.jsp.