Research
Report

An Illustration of the Use of Markov Decision Processes to Represent Student Growth (Learning)

Russell G. Almond

Research &
Development

November 2007
RR-07-40

An Illustration of the Use of Markov Decision Processes to Represent Student Growth (Learning)

Russell G. Almond
ETS, Princeton, NJ

November 2007

As part of its educational and social mission and in fulfilling the organization's nonprofit charter and bylaws, ETS has and continues to learn from and also to lead research that furthers educational and measurement research to advance quality and equity in education and assessment for all users of the organization's products and services.

ETS Research Reports provide preliminary and limited dissemination of ETS research prior to publication. To obtain a PDF or a print copy of a report, please visit:

http://www.ets.org/research/contact.html

Abstract

Over the course of instruction, instructors generally collect a great deal of information about each student. Integrating that information intelligently requires models for how a student's proficiency changes over time. Armed with such models, instructors can *filter* the data—more accurately estimate the student's current proficiency levels—and *forecast* the student's future proficiency levels. The process of instructional planning can be described as a *partially observed Markov decision process* (POMDP). Recently developed computer algorithms can be used to help instructors create strategies for student achievement and identify at-risk students. Implementing this vision requires models for how instructional actions change student proficiencies. The *general action model* (also called the *bowtie model*) separately models the factors contributing to the success or effectiveness of an action, proficiency growth when the action is successful, and proficiency growth when the action is unsuccessful. This class of models requires parameterization, and this paper presents two: a simple linear process model (suitable for continuous proficiencies) and a birth-and-death process model (for proficiency scales expressed as ordered categorical variables). Both models show how to take prerequisites and zones of proximal development into account. The filtering process is illustrated using a simple artificial example.

Key words: Markov decision processes, growth models, prerequisites, zone of proximal development, stochastic processes, particle filter

Acknowledgments

The music tutor example (Section 2) is a free adaptation of an example John Sabatini presented to explain his work in reading comprehension. A large number of details have been changed, partly because John plays guitar while I play wind instruments.

The general action model (Section 4) stems from some consulting work I did with Judy Goldsmith and her colleagues at the University of Kentucky and is really joint work from that collaboration.

Henry Braun asked a number of questions about an earlier draft that helped me better understand the possibilities of the simple filtering techniques, as well as generally improve the clarity of the paper. Frank Rijmen, Judy Goldsmith, Val Shute, Alina von Davier, and Mathias von Davier provided helpful comments on previous drafts. Diego Zapata offered a number of interesting suggestions related to the simulation experiments. René Lawless and Dan Eignor did extensive proofreading and made numerous helpful suggestions to improve the paper's clarity, but should be held blameless with regard to my peculiar conventions (or lack thereof) of English usage and orthography.

Table of Notation

This paper uses the following notational conventions:

- Random variables are denoted by either capital letters (e.g., X), or by words in small capitals (e.g., MECHANICS).
- Variables whose values are assumed to be known are denoted with lower case letters in italics (e.g., t).
- Scaler-valued quantities and random variables are shown in italic type (e.g., X, t), while vector-valued quantities and random variables are put in boldface (e.g., $\mathbf{a}_t$, $\mathbf{S}_t$).
- When a random variable takes on values from a set of tokens instead of a numeric value, then the names of the variable states are underlined (e.g., High, Medium, Low).
- The function $\delta(\cdot)$ has a value of 1 if the expression inside the parenthesis is true and 0 if it is false.
- P$(\cdot)$ is used to indicate a probability of a random event, and $E[\cdot]$ is used to indicate the expectation of a random variable.

Note that Figures 5, 7, and 9 are movies in MPEG-4 format. These should be viewable in recent versions of Adobe Reader, although an additional viewer component may need to be downloaded. If you are having difficulty getting this to work, or if you have a paper copy of this report, the animations may be viewed at `http://www.ets.org/Media/Research/pdf/RR-07-40-MusicTutor.pdf`

Table of Contents

1 Introduction

Black and Wiliam (1998a, 1998b) gathered a number of studies that support the result that teachers using assessment to guide instructional decision-making had measurably better outcomes than teachers who did not. A mathematical model of this decision-making process requires two pieces: a model for the assessment and a model for the instructional activity. Evidence-centered assessment design (ECD; Mislevy, Steinberg, & Almond, 2003) provides a principled approach to developing a mathematical model for the instructional component but not effects of instruction.

Teachers, tutors, and other instructors build qualitative models for the effects of the instruction that become a critical part of their reasoning. ETS's Pathwise® curriculum is typical in this respect. Each unit describes the topic covered—the effect of instruction—and the prerequisites—the factors leading to success. What is typically missing is any quantitative information, information about how likely students are to succeed at the lesson (if the prerequisites are or are not met) and how large the effect is likely to be if the lesson is successful (or unsuccessful).

This report describes a general action model (Section 4; called the *bowtie* model because of the shape of the graph) that is compatible with the qualitative model used informally by instructors. In particular, it provides explicit mechanisms for modeling prerequisite relationships and the effects of actions, as well as providing a general framework for eliciting model parameters from subject-matter experts. The general action model has been used in modeling welfare-to-work counseling (Dekhtyar, Goldsmith, Goldstein, Mathias, & Isenhour, in press; Mathias, Isenhour, Dekhtyar, Goldsmith, & Goldstein, 2006).

One important reason to model the effects of instruction is that it provides a framework for integrating information gathered about a student at multiple points of time (before and after instruction). Consider a typical tutoring regimen. The student and the tutor have regular contacts at which time the student performs certain activities (benchmark assessments) that are designed, at least in part, to assess the student's current level of proficiency. Typically such assessments are limited in time, and hence reliability, but over time the tutor amasses quite a large body of information about the student. However, as

the student's proficiency is presumably changing across time, integrating that information requires a model for student growth.

In many ways the problem of estimating the level of student proficiency is like separating an audio or radio signal from the surrounding noise. Algorithms that use past data to estimate the current level of a process are known in the signal-processing community as *filters*. Section 5 explores the application of filtering techniques in the context of educational measurement.

A related problem to filtering is *forecasting*. Forecasting uses the model of how a proficiency develops to extrapolate a student's proficiency at some future time. Forecasting models have particular value in light of the current emphasis on standards in education. Instructors want to be able to identify students who are at risk for not meeting the standards at the end of the year, in time to provide intervention.

Ultimately, the purpose of the instructor is to form a *plan* for meeting a student's educational goals. A plan is a series of actions—in this case a series of assignments—to maximize the probability of achieving the goal. Instructors need to adapt those plans to changing circumstances and on the basis of new information. In particular, they need to develop a *policy* — rules for choosing actions at each time point based on current estimates of proficiencies.

Understanding what interventions are available is key to building useful assessments. An assessment is useful to a instructor only if that assessment helps the instructor choose between possible actions. Section 3.1 discusses embedding the assessment in the context of the instructor's decision problem. Section 3.2 describes how repeating that small decision process at many time points produces a *Markov decision process*, and Section 3.3 provides a brief review of similar models found in the literature. Casting the instructor's problem in this framework allows us to take advantage of recent research in the field of planning (Section 6).

In order to take advantage of this framework, a model of how proficiencies change over time is needed. Section 4 describes a broad class of models for an *action* that a instructor can take. The general form supports both models where the proficiency variables

are continuous (Section 4.2) and those where they are discrete (Section 4.3). All of these models will be described using a simple music tutoring example introduced in Section 2. Section 5 illustrates this simple example numerically with some simulation experiments.

2 The Music Tutor

Consider a music tutor who meets weekly with a student to teach that student how to play a musical instrument.[1] Each week the tutor evaluates the student's progress and makes an assignment for what the student will do during the next week. To simplify the model, assume that most of the learning occurs during the student's practice during the week. The tutor may demonstrate new concepts and techniques to the student, but it is through using them over the course of the week that the student learns them.

For simplicity, let the domain of proficiency consist of two variables:

- MECHANICS—Being able to find the right fingerings for notes; knowing how to vary dynamics (volume) and articulation; being able to produce scales, chords, trills, and other musical idioms.

- FLUENCY—Being able to play musical phrases without unintended hesitation; being able to sight-read music quickly; playing expressively.

Obviously, there is some overlap between the two concepts, and in a real application better definitions would be needed. Also, although these concepts could increase to arbitrarily high levels (say those obtained by a professional musician), the tutor is only interested in a relatively limited range of proficiency at the bottom of the scale.

Although the proficiencies are correlated, there is another facet of the relationship that must be taken into account: MECHANICS are a *prerequisite* for FLUENCY. For example, if the student has not yet mastered the fingering for a particular note, then the student is unlikely to be able to play passages containing that note without hesitation.

At each meeting between student and tutor, the tutor assigns a practice activity that contains some mixture of exercises and songs (musical pieces or a meaningful part of a musical piece) for the student to practice. Vygotsky's zone of proximal development theory

(Vygotsky, 1978) is clearly relevant to this choice. If the tutor assigns a lesson that is too easy, then the student will not reap much benefit. Similarly, if the tutor assigns a lesson that is too difficult, then very little learning will occur. Part of the problem is assessing the current level of proficiency, so that appropriate lessons can be assigned. Assume that the tutor can adjust the difficulty of the activity in the two dimensions, MECHANICS and FLUENCY. This paper will refer to $\mathbf{a}_t = (a_{t,\text{MECHANICS}}, a_{t,\text{FLUENCY}})$ as the *action* taken by the tutor (or the assignment given by the tutor) at Time t.

Lessons occurs at discrete time points, $t_1, t_2, t_3, \ldots$. In general, these will be regular intervals, but there may be gaps (vacation, missed lessons, etc.). Because what happens between one lesson and the next is of interest frequently, let $\Delta t_n = t_{n+1} - t_n$, the amount of time until the next lesson. (In many applications, Δt_n will be constant across time; however, the notation allows for missed lessons, vacations, and other causes for irregular spacing.) Even in a relatively constrained example, a large number of features of the problem must be modeled. The following section describes those models in more detail.

3 A General Framework for Temporal Models

As seen in the proceeding example, the tutor needs to make not just one decision, but rather a series of decisions, one at each time point. One class of models addressing such a sequence of decisions is the Markov decision process (MDP; Section 3.2). However, before looking at a problem spanning many time points, it is instructive to look at a model with just two time points. Section 3.1 takes a look at this model and the lessons that can be learned from it. The full MDP framework is created by repeating this simple model over multiple time points (Section 3.2). Section 3.3 compares the framework proposed here to others previously studied in the literature. This framework is quite general; specific parameterizations are needed to build models. Section 4 describes one approach to parameterizing these models.

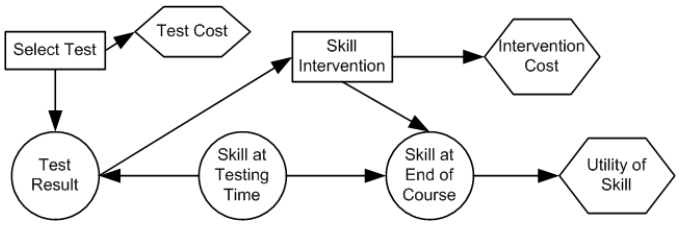

***Figure* 1 Influence diagram for skill training decision.**

Note. An influence diagram that shows factors revolving around a decision about whether or not to send a candidate for training in a particular skill.

3.1 The Value of Assessment

In a typical educational setting, the payoff for both the student and the instructor comes not from the assessment or the instruction but rather from the student's proficiency at the end of the course. Figure 1 shows an *influence diagram* (Howard & Matheson, 1981) illustrating this concept. Influence diagrams have three types of variables represented by three different node shapes:

- Square boxes are *decision variables*. Arrows going into decision variables represent information available at the time when the decision is to be made.
- Circles are *chance nodes* (random variables). Arrows going into the circles indicate that the distribution for a given variable is specified as conditional on its parents in the graph. Note that this sets up the same kind of conditional independence relationships present in Bayesian networks. (Hence an influence diagram consisting purely of chance nodes is a Bayesian network.)
- Hexagonal boxes are *utilities*. Utilities are a way of comparing possible outcomes from the system by assigning them a value on a common scale (often with monetary units). Costs are negative utilities.

On the right of Figure 1 is a node representing the utility associated with a student knowing the skill at the end of the course. The student's probability of knowing the skill at the end of the course depends on both the student's skill level at the beginning of the

course and what kind of instruction the student receives. The instruction has certain costs (both monetary and student's time) associated with it (as does no instruction, but utilities can be scaled so that the cost of no instruction is zero). The student's ability is not known at the beginning of the course, but the student can take a pretest to assess the student's ability. This pretest also has a cost associated with it. The outcome of the pretest can aid the decision about what instruction to give.

The decision of what instruction to provide depends not only on whether or not the student seems to have the skill based on the results of the pretest but also the cost of the instruction and the value of the skill. If the instruction is very expensive and the skill not very valuable, it may not be cost-effective to give the instruction, even if the student does not have the skill. Similarly, the decision about whether or not to test will depend both on the cost of the test and the cost of the instruction. If the instruction is very inexpensive (for example, asking the student to read a short paper or pamphlet), it may be more cost-effective to just give the instruction and not bother with the pretest.

A concept frequently associated with decision analysis problems of this type is the *value of information* (Matheson, 1990). Suppose a perfect assessment could measure the student's initial proficiency without error. Armed with perfect knowledge about the student's initial proficiency, one could make a decision about which instructional action to take and calculate an expected utility. Compare that to the expected utility of the best decision that can be made when nothing is known about the student's initial proficiency level. The difference between these two conditions is the *expected value of perfect information*. In general, real assessments have less than perfect reliability. If the error distribution for an assessment is known, it is possible to calculate its expected value of information for this decision (which will always be less than the value of perfect information). Note that an assessment is only worthwhile when its value of information exceeds its cost.

In many ways, an assessment that has high value of information is similar to the hinge questions of Black and Wiliam (1998a). A hinge question is one used by an instructor to make decisions about how to direct a lesson; in other words, the lesson plan is contingent on the responses to the question. In a similar way an assessment that has high value of

information becomes a hinge in the educational strategy for the student; future actions are based on the outcome of the assessment. Note that there are other ways in which a well-designed assessment can be valuable for learning, such as promoting classroom discussion and helping students learn to self-assess their own level of proficiency.

3.2 Markov Decision Processes

The model of Section 3.1 contains only two time points. The challenge is to extend this model to cover multiple time points, $t_1, t_2, t_3, \ldots$. For simplicity, assume that the cost of testing is small, so that whether to test at each time slice is not an issue. Periodic assessments are sometimes called tracking or benchmark assessments. At each time point, the student is given an assessment and the instructor needs to decide upon an action to carry out until the next time point. Assume that given perfect knowledge of the student's current state of proficiency, the expectations about the student's future proficiency are independent of the student's past proficiency. This Markov assumption allows the periodic assessment and instructional decision process to be modeled as a *Markov decision process* (MDP).[2] The Markov assumption might not hold if there was some latency to the process of acquiring proficiency, or if there was some kind of momentum effect (e.g., having just learned a concept making the student more receptive to learning the next concept). Often a process can be turned into a Markov process by adding another variable to the current state to capture the dependency on the past. In the above example, a *none* could be included for the student self-efficacy or other noncognitive factors that might lead to dependencies to the past.

Figure 2 shows several time slices of an MDP for student learning. Each time slice, $t = 1, 2, 3, \ldots$, represents a short period of time in which the proficiencies of the learner are assumed to be constant. The variables $\mathbf{S}_t$ (vector valued to reflect multiple skills) represent the proficiency of the student at each time slice. The variables $\mathbf{O}_t$ (vector valued to represent multiple tasks or multiple aspects on which a task result may be judged) represent the observed outcomes from assessment tasks assigned during that time slice. The activities are chosen by the instructor and occur between time slices. To model a formative

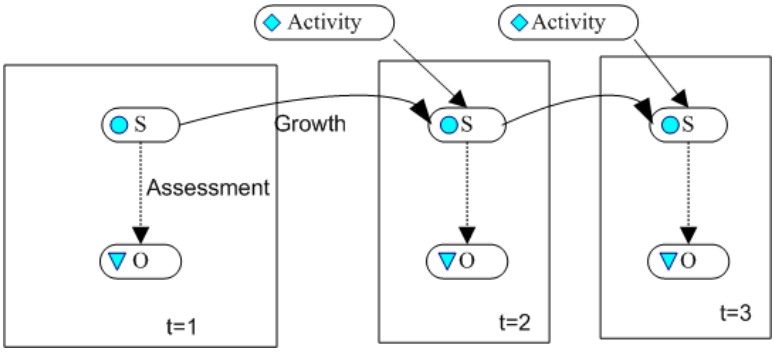

***Figure* 2 Testing as Markov decision process.**

Note. This model shows an MDP made up of two submodels (to be specified later), one for *assessment* and one for *growth*. The nodes marked with circles (◯) represent latent variables, the nodes marked with triangles (▽) are observable quantities, and the nodes marked with diamonds (◇) are decision variables whose quantities are determined by the instructor.

assessment, the time slices could represent periodic events at which benchmark assessments are given (e.g., quarters of the academic year). To model an electronic learning system, the time slices could represent periods in which the system is in *assessment mode*, while the activities represent *instructional tasks*. As a simplifying approximation, assume that no *growth*—change in the state of the proficiency variables—occurs within a time slice.

In general, the proficiency variables **S** are not observable, therefore Figure 2 represents a *partially observable Markov decision process* (POMDP; Boutilier, Dean, & Hanks, 1999). As solving POMDPs is generally hard, opportunities must be sought to take advantage of any special structure in the chosen models for assessment and growth. In particular, the relationship among the proficiencies can be modeled with a Bayesian network.

Dynamic Bayesian networks (DBNs; Dean & Kanazawa, 1989) are Bayesian networks in which some of the edges represent changes over time. (In Figure 3, the curved edges are temporal while the straight edges connect nodes within a single time slice.) The DBN is basically a Bayesian network repeated at each time slice with additional edges connecting (proficiency) variables at one time point to variables at the next time point. Boutilier et

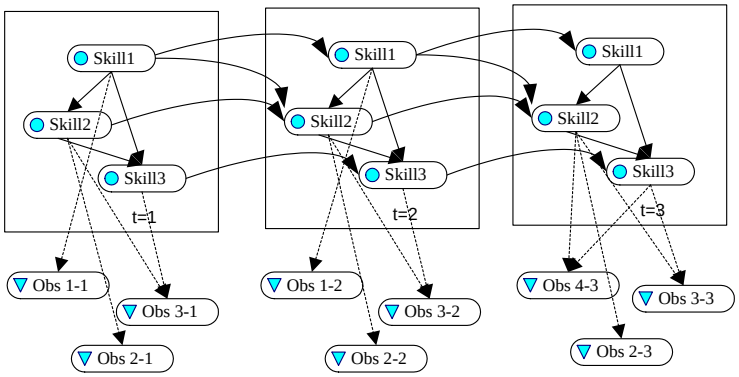

***Figure* 3 Dynamic Bayesian network for education.**

Note. In this figure, the state of the proficiency variables at each time slice is represented with a Bayesian network. The dotted edges represent potential observations (through assessment tasks) at each time slice. The curved edges represent temporal relationships modeling growth among the skills. Note that the assigned tasks and observables may change from time point to time point.

al. (1999) noted that a DBN can be constructed by first constructing a baseline Bayesian network to model the initial time point and then constructing a *two time-slice Bayesian network (2TBN)* to model the change over time.

Although DBNs have been known for over a decade, until recently their computational complexity has made them impractical to employ in an authentic context. Boyen and Koller (1998) presented an approximation technique for drawing inferences from DBNs. Both Koller and Learner (2001) and Murphy and Russell (2001) presented approximations based on particle filtering (Appendix A; Liu, 2001; Doucet, Freitas, & Gordon, 2001). Takikawa, D'Ambrosia, and Wright (2002) suggested further efficiencies in the algorithm by noting variables that do not change over time, producing partially dynamic Bayesian networks.

There is a close correspondence between the POMDP model of Figure 2 and evidence-centered design objects (Mislevy et al., 2003). The nodes $\mathbf{S}_t$ represent the proficiency model and the nodes $\mathbf{O}_t$ (or more properly the edges between $\mathbf{S}_t$ and $\mathbf{O_t}$) represent the evidence models. What this framework has added are the *action models*: the

2TBNs representing the change between $\mathbf{S}_{t_1}$ and $\mathbf{S}_{t_2}$. One of the principal challenges of this research is to build simple mathematical models of proficiency growth (the 2TBNs) that correspond to the cognitive understanding of the process. The term *action model* recognizes that actions taken by the instructor will influence the future proficiency level of the student.

3.3 Similarity to Other Temporal Models

Removing the activity nodes from Figure 2 produces a figure that looks like a hidden Markov model, a class of models that has found applications in many fields. Langeheine and Pol (1990) described a general framework for using Markov models in psychological measurement. Eid (2002) used hidden Markov models to model change in consumer preference; and Rijmen, Vansteelandt, and De Boeck (in press) used these models to model change in patient state at a psychiatric clinic. Both of these applications have a significant difference from the educational application in that the set of states is unordered (consumer preference), or at best partially ordered, while there is a natural order to the educational states. Generally, in education the student moves from lower states to higher states, where in the other applications the patient can move readily between states. Reye (2004) looked at Markov models in the context of education and also arrived at a model based on dynamic Bayesian networks.

Similar models for change in student ability have been constructed using latent growth curves, structural equation models and hierarchical linear models (Raudenbush, 2001). Von Davier and von Davier (2005) provided a review of some of the current research trends and software used for estimating these models. One complication addressed in many of these models, but not in this paper, is the hierarchy of student, classroom, teacher, and school effects, all of which can affect the rate at which students learn.

If the tests administered at each time point are not strictly parallel, then the issue of equating the test forms, sometimes called vertical scaling, arises. As this goes far beyond the scope of this paper, which does not address estimation in depth, all of the benchmark tests are assumed to be either strictly parallel or have been placed on a common vertical scale.

Most of the research surveyed in the papers mentioned above addresses actions in the context of attempting to measure the effectiveness of a particular intervention. This is a difficult problem, principally because the sizes of the instructional effects are often smaller than the student-to-student and/or classroom level variability, and instructional treatments are often applied at the classroom level. Note that causal attribution is not necessary for the MDP, that is, it is not necessary to separate the contributions of the teacher, the class, and the curriculum but rather to have a good estimate of the total effect. In the classical context of program evaluation, current educational practice is usually given priority: A new program must show that it is sufficiently better and that the improvement can be noticed above the measurement error and between subject variability. In the decision theoretic framework, priority is given to the method with the lowest cost (often, but not always, the standard treatment). If the actions have equal cost, decision theory favors the action with the highest expected effect, even if that difference is not statistically significant (i.e., the methods have roughly equal value).

The key difference between the approach outlined in this report and others discussed above is that the MDP framework treats instruction as an action variable to be manipulated, rather than an observed covariate. This requires an explicit model for how an action affects student proficiency. Section 4 develops that model.

4 General Action Model

Let $\mathbf{S}_t = (S_{1,t}, \ldots, S_{K,t})$ be the student's proficiencies on K different skills at time t. Under the assumptions of the MDP, only the current value of the proficiency variables, $\mathbf{S}_{t_1}$ and the action chosen at Time t_1, $\mathbf{a}_{t_1}$ are relevant for predicting the state of of the proficiency variables at the next time point, $\mathbf{S}_{t_2}$. A general assumption of the MDP framework is that the effects of actions depend only on the current state of the system (the student's current proficiency profile). Further, assume that each potential action, $\mathbf{a}$, has a focal skill, $S_{k(\mathbf{a})}$, that it targets. Under these conditions, the MDP is specified through a set of transition probabilities $P_{\mathbf{a}}(S_{k(\mathbf{a}),t_2}|\mathbf{S}_{t_1})$ describing the possible states of the focal skill at the next time point given all of the skills at the current time point. (An action can have

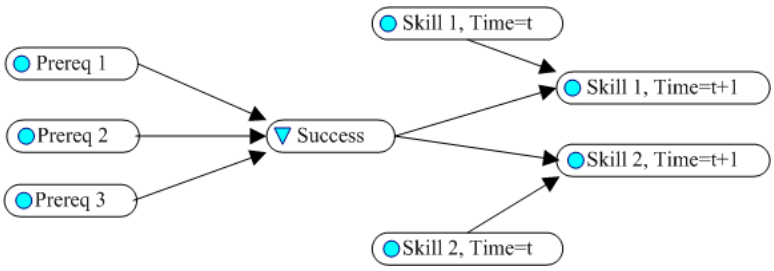

***Figure* 4 A general action model.**

Note. The central node is the students success or failure in the assigned activity. This is influenced by the current values of certain prerequisite skill nodes. The transition probabilities are influenced by the prerequisites only through the success or failure of the action. Success may or may not be observed.

multiple focal skills, in which case the transition probabilities become a joint distribution over all focal skills.)

One factor that affects the transition probabilities is how successfully the student completes the assigned work. To simplify the model, it is assumed that the outcome from all actions can be represented with a binary variable indicating successful completion. (The state of the SUCCESS variable may or may not be observable.) The probability of success for an activity will depend on the state of prerequisite skill variables and possibly on student characteristics (i.e., the student's receptiveness to various styles of instruction). The Markov transition probabilities for various skill variables will depend on the success of the activity (with different transition probabilities for success and failure).

Figure 4 shows a generic action model. Mathias et al. (2006) called this model a *bowtie* model because the left and right halves are separated by the *knot* of the SUCCESS variable. It is generally assumed that an instructional activity can be either successful or not. Certain prerequisite skills will influence the probability of a successful outcome. Successful (and unsuccessful) outcomes will each have certain probabilities of changing one of the underlying proficiencies.

This model contains a key simplification: The value of each focal skill (a skill targeted

by instruction) at Time $t + 1$ is conditionally independent from the prerequisites given the SUCCESS of the action and the value of the skill at Time t. Effectively, SUCCESS is a switch that selects one of two different sets of transition probabilities for each focal skill. Furthermore, interaction between skills occurs only through the prerequisites for SUCCESS (note that the focal skill could appear both on the left-hand side of the knot as a prerequisite and on the right-hand side as the old value of the skill). This implies that the model needs only two sets of transition probabilities for the focal skills of the action, instead of a set of transition probabilities for each configuration of the prerequisites. If the action has multiple focal skills, then the change in each focal skill is assumed to be independent given proficiency. (In more complex situations, a separate bowtie could be built for each focal skill).

The simplification provided by the bowtie is also very important when working with experts to elicit model structure and prior values for model parameters. Under the bowtie model, experts need only worry about interactions among the skills when describing the prerequisites for SUCCESS of an action. When defining the effects of the action, the experts can work one skill at a time. This simplification may not result in models that accurately reflect of the cognitive process, but it should result in models that are tractable to specify and compute.

This model was originally developed in collaboration with Judy Goldsmith and colleagues at the University of Kentucky who were developing a welfare-to-work advising system (Dekhtyar et al., in press; Mathias et al., 2006). The counseling problem has a longer time scale than the music example. It also has some actions for which the success may be observed (e.g., completing a high school GED program) and some for which it may not be observed (e.g., the success of a substance abuse treatment program). Mathias et al. (2006) found that experts (welfare case workers) had difficulty specifying a complete set of transition probabilities for an action, but did feel comfortable supplying a list of prerequisites and consequences of an action along with a relative indication of the strength and direction for each variable.

Vygotsky's zone of proximal development for an action can be modeled in both the

probability of success and the transition probabilities. Assignments that are too hard are modeled with a low probability of success. This is reflected in the model by making the focal proficiency a prerequisite for success. Assignments that are too easy are marked by reducing the transition probability to higher states. If such an assignment is completed successfully, the probability of moving from a low proficiency state to a middle state is high, but the probability of moving from a middle to a high proficiency state is low.

The general action model makes a number of crude assumptions, but it still has some attractive properties. In particular, it has a very clean graphical structure and supports probability distributions containing only a few parameters related to concepts with which experts will resonate. More importantly, it splits the problem of modeling an action into two pieces: modeling the factors contributing to success (Section 4.1) and modeling the effects of the action given success (Section 4.2 for continuous proficiency variables and Section 4.3 for discrete). Each piece can then be addressed separately.

4.1 The Noisy-And Model of Success

The SUCCESS variable in the bowtie model renders the left-hand side (modeling the prerequisite relationships) and the right-hand side (modeling student growth) conditionally independent. Sections 4.2 and 4.3 describe the models for growth. The key contribution of the SUCCESS variable is that it allows for two different growth curves, one for when the lesson was effective and one for when it was ineffective.

Let X_t represent the value of the SUCCESS variable for the action assigned at Time t, coded $\underline{1}$ for success and $\underline{0}$ for failure. Here *success* lumps together motivation—did the student do the lesson—mastery of the proficiency addressed in the lesson—did the student do it right — and appropriateness—was this lesson an effective use of the student's practice time. The left-hand side of the bowtie model requires that the distribution of success must be given conditioned on the values of the prerequisite proficiency variables at the time of the start of the lesson. In the most general case, the number of parameters can grow exponentially with the number of prerequisites; therefore, it is worth trying to find a simpler model for this part of the relationship.

Usually, for an action to be successful all the *prerequisites* for that action must be met. For each prerequisite, $S_{k'}$, let $s^{-}_{k',\mathbf{a}}$ be the minimum level of that proficiency required for Action $\mathbf{a}$ to have a high probability of success, and let $\delta(S_{k',t} \geq s^{-}_{k',\mathbf{a}}) = 1$ if $S_{k',t} \geq s^{-}_{k',\mathbf{a}}$ is true (student meets prerequisites for Proficiency k') and $\delta(S_{k',t} \geq s^{-}_{k',\mathbf{a}}) = 0$ otherwise. Even given the current proficiency level of the student, some uncertainty remains about the success of the action, X_t. In particular, let $q_{k',\mathbf{a}}$ be the probability that a student assigned Action $\mathbf{a}$ compensates for missing the prerequisite, $S_{k'}$, and let $q_{0,\mathbf{a}}$ be the probability that a student who meet the prerequisites successfully completes the activity. Then the *noisy-and* distribution (Junker & Sijtsma, 2001; Pearl, 1988) can model the relationship between $\mathbf{S}$ and X:

$$\mathrm{P}(X_t = 1|\mathbf{S}_t, \mathbf{a}) = q_{0,\mathbf{a}} \prod_{k'} q_{k',\mathbf{a}}^{1-\delta(S_{k',t} \geq s^{-}_{k',\mathbf{a}})} \ . \tag{1}$$

Note that zone of proximal development considerations could be added by making the interval two-sided (e.g., $\delta(s^{+}_{k',\mathbf{a}} \geq S_{k',t} \geq s^{-}_{k',\mathbf{a}})$). This would state that the action would usually be unsuccessful (ineffective) if the prerequisite skill was either too high or too low. The use of bounds also conveniently renders all variables as binary, which is necessary for the noisy-and model.

The number of parameters in this model is linear in the number of prerequisite skills. The experts must specify (or the analysts must learn from data) only the lower and upper bound for each prerequisite and the probability that the student can work around the missing prerequisite, $q_{k,\mathbf{a}}$. In addition, the probability that the lesson will be successful when the prerequisites are met, $q_{0,\mathbf{a}}$ must be specified. In practice, experts will generally specify the thresholds and prior distributions for the $q_{k,\mathbf{a}}$ parameters. The latter can be refined through calibration, if appropriate longitudinal data are available. This seems like a large number of parameters to work with; however, without some kind of modeling restriction, the number of parameters grows exponentially with the number of prerequisites. Furthermore, the boundaries should be close to the natural way that experts think about instructional activities (i.e., this activity is appropriate for persons whose skill is in this range and who meets these prerequisites).

One interesting possibility for this model is to explain how instructional scaffolding might help in an action. Presumably, scaffolding works to lower the level of the prerequisite skills required for a high chance of success and thus raises the probability of success for an action that is unlikely to be effective without scaffolding.

Note that the value of the SUCCESS probability can be observed or unobserved. When SUCCESS is observed at Time t, it also provides information about the proficiency of the student at Time t. SUCCESS is more likely at some proficiency states than others. Hence, it can be treated as an additional observable at Time t. When SUCCESS is not observed, there may be some indirect measure of success, such as the student's self-report. With a little more work, these measures can be incorporated into the model. Also, with a little more work, there could be multiple values for SUCCESS (for example, the grade received on an assignment). This might be useful for modeling situations in which there are many different growth curves in the population. However, the number of parameters increases as the number of states of SUCCESS increases, on both the right- and left-hand sides of the bowtie model.

4.2 Linear Growth Model

Assuming that the underlying proficiencies are continuous and that growth occurs in a linear way (albeit with noise) leads to a linear growth model. This model is similar to one used in a classical signal processing problem called the Kalman filter (Kalman & Bucy, 1961).

Following the general convention for the Markov decision process framework, a model is needed for the initial timeslice and then a model for what happens at each increment. For the initial conditions, a multivariate normal distribution is assumed: $\mathbf{S}_0 \sim N(\theta_0, \mathbf{\Sigma}_0)$.

At this point, a fundamental assumption is made: Practice increases skills and skills decrease with lack of practice. According to this assumption, the growth model can be split into two pieces. In the background, skills deteriorate over time unless that decrease is offset by practice. If the skill is being actively and effectively practiced, then the skill will increase over time. However, the effectiveness of the practice depends on how well the

selected assignment (the action) is matched to the student's current ability levels.

First, the background deterioration of the proficiency is examined. That the proficiency deteriorates at a fixed rate over time is assumed. Let μ_k be the deterioration rate for proficiency S_k. In the absence of instruction,

$$E[S_{k,t_2}|S_{k,t_1}] = S_{k,t_1} - \mu_k \Delta t_1 \ .$$

Note that if the skill is normally practiced in day-to-day life, then μ_k might be zero or negative (indicating a slow background growth in the skill).

Instruction, on the other hand, should produce an increase in the skill at the rate $\lambda_k(\mathbf{a}_t, \mathbf{S}_t, X_t)$, which is a function of the current proficiency state, $\mathbf{S}_t$, the action taken at Time t, $\mathbf{a}_t$ and the success of that action, X_t. The correct time line for proficiency growth is rehearsal time and not calendar time. However, in most practical problems, the rehearsal time will be a multiple of Δt, possibly depending on the action and its success. Therefore, the difference between rehearsal and calendar time can be built into the growth rate, $\lambda_k(\mathbf{a}_t, \mathbf{S}_t, X_t)$. Thus, the proficiency value at time t_2 will be:

$$S_{k,t_2} = S_{k,t_1} + \lambda_k(\mathbf{a}_t, \mathbf{S}_t, X_t)\Delta t_1 - \mu_k \Delta t_1 + \epsilon_{t,k} \ , \tag{2}$$

where $\epsilon_{t,k} \sim N(0, \Delta t \sigma^2_{\mathbf{a}})$. Making the variance depend on the elapsed time is a usual assumption of Brownian motion processes (Ross, 1989).

As stated, neither $\lambda_k(\mathbf{a}_t, \mathbf{S}_t, X_t)$ nor μ_k are identifiable from data. One possible constraint is to set a zero growth rate for all unsuccessful actions, $\lambda_k(\mathbf{a}_t, \mathbf{S}_t, X_t = 0) = 0$. This models a common rate of skill growth (deterioration), μ_k, for failure across all actions. This is appealing from a parameter reduction standpoint. Another possibility is to set μ_k to a fixed value, thus effectively modeling the proficiency change when the action fails individually for each action.

As $\lambda_k(\mathbf{a}_t, \mathbf{S}_t, X_t)$ depends on X_t, it already incorporates the prerequisite relationships and zone of proximal development; however, it does so through the use of hard boundaries. Although Equation 1 captures the prerequisite relationships, it does not do a good job of capturing the zone of proximal development. An assignment that is a little too easy is

expected to have some value, just not as much. The same will be true for an assignment that is a little too hard. On the other hand, an assignment that is much too easy or much too hard should produce little benefit. Let $s^*_{k,\mathbf{a}}$ be the target difficulty for Action $\mathbf{a}$. One possible parameterization for, $\lambda_k(\mathbf{S}_t, \mathbf{a}, X_t)$ is:

$$\lambda_k(\mathbf{S}_t, \mathbf{a}, X_t) = \eta_{k,\mathbf{a},X_t} - \gamma_{k,\mathbf{a},X_t}(S_{k,t} - s^*_{k,\mathbf{a}})^2 \ . \tag{3}$$

The first term, $\eta_{k,\mathbf{a},X_t}$, is an action specific constant effect. The second term is a penalty for the action being at an inappropriate level; the parameter $\gamma_{k,\mathbf{a},X_t}$ describes the strength of this penalty. The choice of squared error loss to encode the proximal zone of development is fairly conventional. It has the desired property of having zero penalty when the student is exactly at the target level for the action with the penalty going up rapidly as the student's proficiency level is either too high or too low. In fact, it may be necessary to truncate Equation 3 so that $\lambda_k(\mathbf{S}_t, \mathbf{a}, X_t)$ never falls below zero. One possible problem with this parameterization of λ is that it treats insufficient and excess skill symmetrically. This may or may not be realistic. However, using the focal proficiency as both a prerequisite and in the proximal zone equation allows limited modeling of asymmetric relationships.

Two values of η and γ must be specified for each action, $\mathbf{a}$, and each proficiency variable affected by the action, S_k: one set of values is needed for a successful action, $X_t = 1$, and one for an unsuccessful action, $X_t = 0$. As noted above, in many situations it may be desirable to set the values of η and γ to zero when the action is not successful and let the base rate μ_k dominate the change in ability.

4.3 Birth-and-Death Process Model

The insights gained from the linear model can be used to create a similarly simple model for discrete proficiencies. For this section, assume that the proficiency variables range over the values 0 to 5 [3] (truncating the proficiency scale at the higher abilities model students graduating from this tutoring program).

Assume that the time interval Δt_n is always small enough that the probability of a student moving more than one proficiency level between lessons is negligible. In this

case, a class of stochastic process models called the *birth-and-death process* (Ross, 1989) is often used. It is called a birth-and-death process because it is useful for modeling population change. In this example, the student's ability corresponds to the population. The birth-and-death process assumes that birth (increase) and death (decrease) events both happen according to independent Poisson processes. In the case of skill acquisition, the birth rate is the rate of skill growth, $\lambda_k(\mathbf{S}_t, \mathbf{a}, X_t)$, and the death rate, μ_k, is the rate of skill deterioration.

Actually, this is not quite a classic birth-and-death process because there is an absorbing state at the top of the ability scale. In some ways, this makes things easier because the outcome spaces are finite (and the same model is not likely to be valid for infinite ability ranges). A second difference from the classical birth-and-death model is that the birth rate is a random variable (as it depends on the random success, X_t, of the action).

In general, both the improvement and deterioration rate will depend on the student's current proficiency level. At the minimum, the rates at the boundary conditions will be zero (a student cannot transition lower from the lowest state or higher than the highest). It seems sensible to keep the deterioration rate, μ_k, constant, except at the boundary. Similarly, Equation 3 could be pressed into service to generate the improvement rates.

5 Examples of the Models in Action

A small simulated data experiment should aid in understanding how the proposed temporal models can unfold. Section 5.1 describes the parameters used in simulation, based on the Music Tutor example (Section 2). Section 5.2 looks at a couple of simulated series and at how well they can be estimated by various techniques. Section 5.3 summarizes key findings from the simulation.

5.1 Music Tutoring Simulation

The music tutoring problem defines two proficiency scales, MECHANICS and FLUENCY. The student's proficiencies on these two scales at Time t, taken together, form $\mathbf{S}_t$. To further specify the scale for a numerical example, both are continuous and take on values

between 0 and 6. To further define the scale, assume that the the student is assumed to be randomly selected from a population with the following normal distribution:

$$\mathbf{S}_0 \sim N(\boldsymbol{\mu}, \Omega)$$

where

$$\boldsymbol{\mu} = \begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix} \quad , \quad \text{and} \quad \Omega = \begin{bmatrix} 0.52 & 0.48 \\ 0.48 & 0.52 \end{bmatrix} \quad .$$

Assume that at each meeting with the student, the tutor evaluates the student's current proficiency rating, assigning a score for MECHANICS and FLUENCY based on the same scale as the underlying proficiencies (0–6), but rounded to the nearest half-integer. Assume that the assessment at any time point, $\mathbf{Y}_t$, has relatively low reliability and that the two measures are confounded so that the MECHANICS proficiency influences the FLUENCY component of the score and vice-versa. The evidence model for these benchmark evaluations is given by the following equation:

$$\mathbf{Y}_t = A\mathbf{S}_t + \mathbf{e}_t \ , \tag{4}$$

where

$$\mathbf{e}_t \sim N(\mathbf{0}, \Lambda) \ ,$$

with

$$A = \begin{bmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{bmatrix} \quad , \quad \text{and} \quad \Lambda = \begin{bmatrix} 0.65 & 0 \\ 0 & 0.65 \end{bmatrix} \quad ,$$

and the tutor rounds the scores to the nearest half-integer.

The numbers stated above are enough to calculate the reliability of the benchmark assessment at Time 0. The expression $\Omega_{ii}/(\Omega + A^{-1}\Lambda(A^{-1})^T))_{ii}$, gives the reliability for the measurement of Skill i. Using the number from the simulation, this gives a reliability of 0.45 at Time 0.

As time passes, however, the variance of the population will increase. Equation 2 describes a random walk in discrete time, or a Brownian motion process. In both cases the variance of $S_{k,t}$ should be $\sigma^2_{S_{k,0}} + t\sigma^2_{\Delta S_k}$, where $\sigma^2_{S_{k,0}}$ is the Time 0 population variance and

$\sigma^2_{\Delta S_k}$ is a quantity that describes the variance of the innovations (the result of differencing Equation 2) at each time step. Thus, both the numerator and denominator in the reliability equation will increase over time, and the reliability will increase. This is a reflection of a well-known fact that reliability is population dependent.

In general, the variance of $\Delta \mathbf{S}_t$ (change in skill from time to time) is difficult to calculate, in part because it depends on the policy—the series of rules by which actions are chosen. Thus, in the model described here, there are three sources of variance for the difference in skill between time points: (a) the random response to the chosen action (as expressed by the SUCCESS variable); (b) estimation error for the current proficiency level that causes the chosen action to target a proficiency higher or lower than the actual one (thus affecting the growth rate); and (c) the variation of the observed growth around the average growth curve, the quantity referred to as $\epsilon_{k,t}$ in Equation 2.

For the tutor, choosing an action, $\mathbf{a}_t$, consists of choosing a MECHANICS and FLUENCY level for the assignment. (Ignore for the moment the fact that lessons with high FLUENCY and low MECHANICS demands might be difficult to obtain, as they will rarely be called for by the model.) A reasonable first guess at an optimal policy is for the tutor to simply assign a lesson based on the tutor's best estimate of current student proficiency.

Using the bowtie model, the action is modeled in two pieces: one for the success of the action and one for the effect of the action given the success status. In this simulation, each proficiency dimension is assigned a separate success value, X_{tk}, taking on the value $\underline{1}$ if the lesson is effective and $\underline{0}$ if the lesson is not effective. Furthermore, the two success values are assumed to be independent given the proficiency. A key component of the model is that the proficiency level of the assigned lesson, $\mathbf{a}_t$, must be within the zone of proximal development of the student's proficiency, $\mathbf{S}_t$. This is accomplished by setting bounds for $S_{k,t} - a_{k,t}$.

For the MECHANICS skill, there is still some benefit from practicing easy techniques (large positive difference between skill and action), but if the technique to be practiced is much above the student's current MECHANICS proficiency (negative difference), the benefit will be minimal. For that reason, the MECHANICS prerequisite is considered met

when $\delta_M(\mathbf{S}_t, \mathbf{a}_t) = \delta(-1 \leq S_{\text{MECHANICS},t} - a_{\text{MECHANICS},t} \leq 2)$. For FLUENCY the opposite is true; rehearsing musical pieces below the current student level has minimal benefit, while attempting a piece that is a musical stretch is still worthwhile. Therefore, the FLUENCY prerequisite is considered met when $\delta_F(\mathbf{S}_t, \mathbf{a}_t) = \delta(-1 \leq S_{\text{FLUENCY},t} - a_{\text{FLUENCY},2} \leq 2)$. Because students may benefit from the lesson even if the prerequisites are not met, *guessing parameters*, $\mathbf{q}_1$, are added to the model. For this example, set $q_{1,\text{MECHANICS}} = .4$ and $q_{1,\text{MECHANICS}} = .5$, reflecting the fact that mechanical ability is harder to fake than fluency.

Finally, Equation 1 contains a parameter, q_0, for the probability that a student succeed given that the prerequisites are met. As the success variable is two-dimensional, this parameter is two-dimensional as well. The probabilities are .8 for MECHANICS and .9 for FLUENCY, reflecting the fact that students are usually less motivated to study mechanics than fluency. Multiplying the values of $\mathbf{q}$ and q_0 out according to Equation 1 yields the conditional probability table for $\text{P}(\mathbf{X}_t|\mathbf{S}_t, \mathbf{a}_t)$, shown in Table 1.

Table 1

Success Probabilities by Prerequisite

$\delta_M(\mathbf{S}_t, \mathbf{a}_t)$	$\delta_F(\mathbf{S}_t, \mathbf{a}_t)$	$\text{P}(X_{t,\text{MECHANICS}} = 1)$	$\text{P}(X_{t,\text{FLUENCY}} = 1)$
TRUE	TRUE	.80	.90
TRUE	FALSE	.40	.45
FALSE	TRUE	.32	.36
FALSE	FALSE	.16	.18

The relatively simplistic assumption can be made that successful practice is 10 times as effective as unsuccessful practice. Thus, it is possible to simply multiply the rate constant λ by $(.9 * X_t + .1)$ to get the effective improvement rate for each time period.

Setting the improvement and deterioration rates requires first picking a time scale. If the base time scale is months, then a monthly series can be simulated by using a time step of 1 between data points and a weekly series by using a time step of 1/4. Using this scale, the MECHANICS proficiency deteriorates at the rate of .02 proficiency units per month,

while FLUENCY deteriorates at the rate of .01 proficiency units per month. Thus, over a short simulation, proficiency is likely to stay fairly level with no instruction.

Recall that according to Equation 3, the improvement rate has two components: a constant term η and a quadratic penalty term, γ, for being far from the target proficiency of the lesson. For both proficiencies, η was set at .2 proficiency units per month (10–20 times the deterioration rate), while γ was set to .1 for MECHANICS and .15 for FLUENCY, indicating that mechanical practice for a lesson that is off target provides more benefit than fluency practice (provided that the lesson was effective).

Naturally, there is some random variability in the proficiency change at each time point. Assume that this happens according to a Brownian motion process (see Ross, 1989), so that the variance will increase with the elapsed time between lessons. In this case, $\mathrm{Var}(\epsilon_{k,t}) = \sigma^2 \Delta t$. For this example, σ is set to .02 for both proficiencies. (Note that the model does not require that the errors be uncorrelated across the dimensions; the error could instead be described in terms of a covariance matrix.)

Using these numbers and a policy for making assignments at each time point, it is reasonably straightforward to write a simulation. Appendix B gives the code written in the R language (R Development Core Team, 2005) used to perform the simulation. Two questions are of immediate interest: (a) given a sequence of observations, what is the most likely value for the underlying student proficiency? and (b) how can a policy be set for assigning an action at each time point?

The first corresponds to a classic time series concept called *filtering*. The name comes from signal processing, in which high frequency noise (measurement error) is separated from the underlying signal. Filters take the previous observations as input and output an estimate of the current level of the signal (or in this case, the current proficiency level). Appendix A describes several possible filters.

One filter that is able to take advantage of the structure and constraints of the problem is the *particle filter* (Doucet et al., 2001; Liu, 2001). The particle filter works by generating a number of plausible trajectories for the student through proficiency space and weighting each one by its likelihood of generating the sequence of observed outcomes (Appendix A)

provides more detail). This can be represented graphically (Figure 5) by a cloud of points colored with an intensity related to their weights. The best estimate for the student at each time point is the weighted average of the particle locations at that time point. A 95% credibility interval can be formed by selecting a region that contains 95% of the particles by weight.

Although the particle filter is quite flexible, it is also computationally intensive. For the purposes of the simulation, the policy was based on an ability estimate that was simpler to compute. In the simulation described below, the tutor's assignments were based on an estimate using an *exponential filter*, a simple weighted sum of the current observation and the previous estimate. Exponential filters are optimal when the underlying process is well approximated by an integrated moving average process (Box & Jenkins, 1976), a simple class of models that often provides a good approximation to the behavior of nonstationary time series. Appendix A describes the exponential filter. The weight is set rather arbitrarily at 0.5.

Finally, the policy needs to be specified. In this case, the policy is straightforward: the instructor targets the lesson to the best estimate (using the exponential filter) of the student's current ability.

5.2 The Simulation Experiment

Using the model of Section 5.1, assume that the student and music tutor meet monthly. Using this basic framework, Table 2 shows data for a simulated year of interactions between tutor and student. Note that the actions shown in Table 2 are based on matching the action level to the average of the previous three observed values (rounded to the nearest half-integer). This provides an opportunity to compare the particle filter, which incorporates the model for growth, to the much simpler exponential filter, which does not.

Figure 5 is an animation[4] of the result of applying the particle filter to the data in Table 2, with the value of the success variable assumed to be observed (input to the filter). The diamond represents the student simulated proficiency. This is the ground truth of the simulation and the target of inference. The triangle plots the noisy (low reliability)

Table 2

Monthly Simulated Data for Music Tutor Example

Time	Truth		Observed		Action		Success	
Months	Mech.	Flu.	Mech.	Flu.	Mech.	Flu.	Mech.	Flu.
0	1.55	1.37	1.5	1.5	1.5	1.5	1	1
1	1.73	1.51	2.0	1.5	1.5	1.5	1	1
2	1.88	1.72	2.5	1.0	2.0	1.0	1	1
3	2.05	1.84	3.0	2.0	2.5	1.5	1	1
4	2.21	2.02	2.0	2.0	2.0	1.5	1	1
5	2.39	2.15	1.0	2.0	1.5	1.5	1	1
6	2.48	2.26	2.5	3.0	2.0	2.0	1	1
7	2.65	2.42	4.0	3.0	3.0	2.5	1	1
8	2.79	2.59	2.5	1.0	2.5	1.5	0	0
9	2.79	2.59	3.0	3.0	2.5	2.0	0	1
10	2.78	2.73	3.5	2.0	3.0	2.0	0	1
11	2.77	2.84	3.0	3.0	3.0	2.5	1	1
12	2.97	3.00	2.5	3.5				

benchmark test. The star plots the estimate from the exponential filter, which the simulated tutor uses to decide on the action.

The estimate from the particle filter is shown with the cloud of circles. The circles are colored gray with the intensity proportional to the weights. (The gray value[5] is $1 - w_t^{(n)}/max_n w_t^{(n)}$, where $w_t^{(n)}$ is the weight assigned to particle n at time t. The point with the highest weight is black, and the other particles are increasingly paler shades of gray.) In order to avoid a large number of particle with low weights, the collection of particles is periodically resampled from the existing particles (this is called the *bootstrap filter*; see Appendix A).

The performance of the estimators can be seen a little more clearly by looking at the

http://www.ets.org/Media/Research/pdf/RR-07-40-Sime2m.mp4

***Figure* 5 Particle filter estimates for Music Tutor example, action success observed.**

estimates for the two proficiency variables as separate time series. Figure 6 shows the same estimates as Figure 5 from a different orientation. The upper and lower bounds for the particle filter estimate are formed by looking at the value at which 97.5% (or 2.5%) of the weight falls below. Note that although the true value moves to the upper and lower bounds of the estimate, it always stays within the interval. The exponential filter smoothes some but not all of the variability. The amount of smoothing could be increased by decreasing the weight in the filter; however, this will make the estimates more sensitive to the initial value.

It may or may not be possible to observe the success of the monthly assignments. Figures 7[6] and 8 show the results of the particle filter estimate when the action is not observed. Note that the 95% interval is wider on the new scale.

By changing the value of Δt, the simulation can be run for weekly rather than monthly

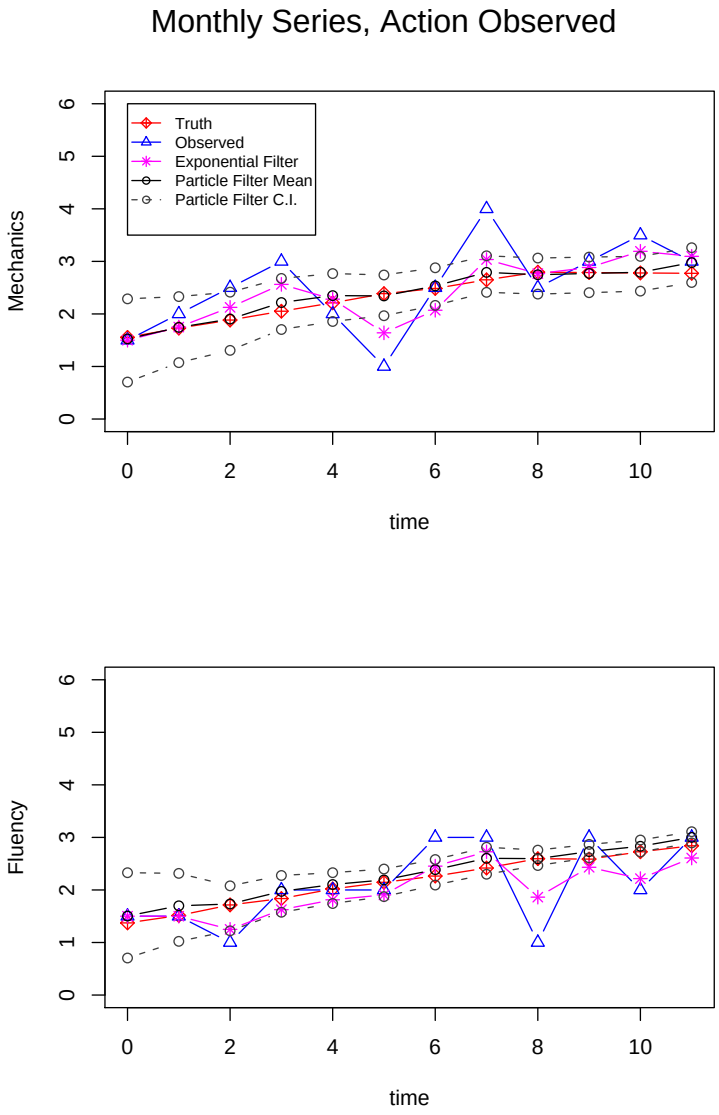

***Figure* 6 Time series estimates for Music Tutor example, action success observed.**

http://www.ets.org/Media/Research/pdf/RR-07-40-Sime2mna.mp4

***Figure* 7 Particle filter estimates for Music Tutor example, action success unobserved.**

meetings between tutor and student. In the weekly scheme, the tutor has the ability to adjust the lesson on a weekly basis; there are also more observations. Figures 9 [7] and 10 show the results. Again, the particle filter tracks the true proficiency to within the 95% confidence band, although in many cases it looks like the true value lies close to the lower limit.

5.3 Analysis of This Example

The particle filter technique looks good in these artificial settings, where the simulation model matches the model used in the particle filter. With this choice of weight, the exponential filter smooths out only a small part of the noise of the series. Decreasing the weight results in a smoother estimates increasing the reliability of the benchmark assessment should produce an increase in the smoothness of both the exponential and

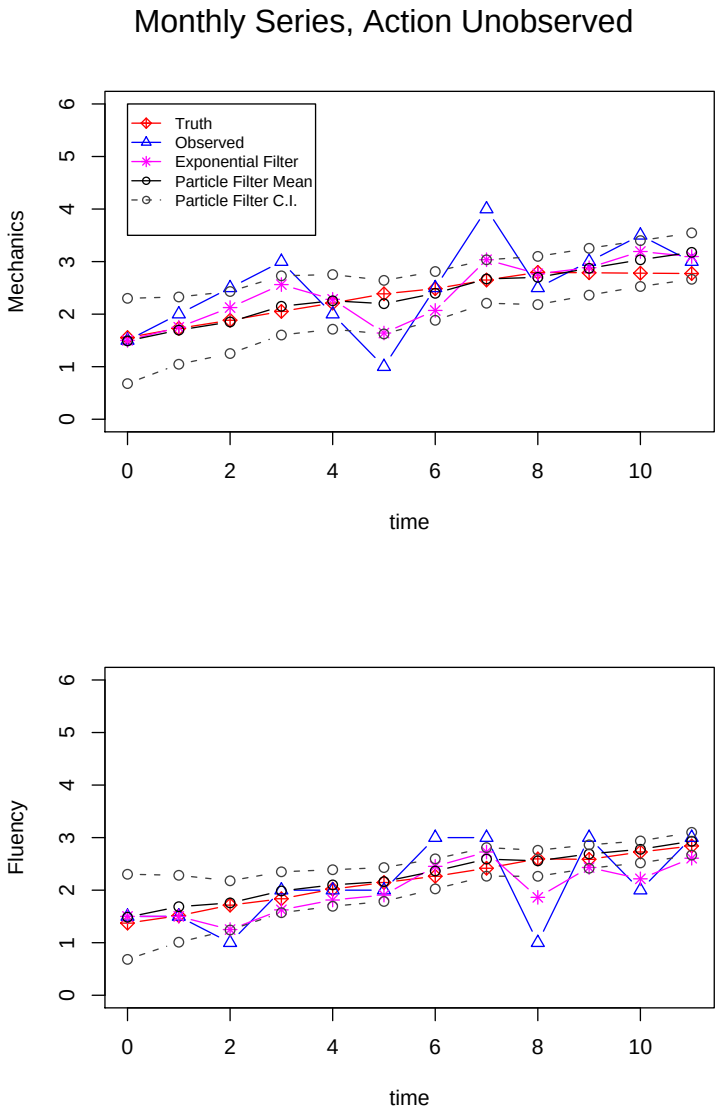

***Figure* 8 Time series estimates for Music Tutor example, action success unobserved.**

http://www.ets.org/Media/Research/pdf/RR-07-40-Sime2wna.mp4

***Figure* 9 Particle filter estimates for Music Tutor example, action success unobserved, weekly meetings.**

particle filter estimate. The effect should be stronger in the exponential filter, which is not making optimal use of past observations to stabilize the estimates. Similarly, if the variance of the skill growth process is large, the particle filter will have less information from past observations with which to stabilize the current estimates. Thus, its improvement over the exponential filter type estimator will be more modest.

Note that the benchmark assessment was deliberately chosen to have low reliability. The practical considerations of time spent in the classroom for instruction versus assessment require that the benchmark assessments be short in duration. That means that as a practical matter the reliability of the benchmark assessments will be at best modest.

A larger problem with the exponential filter is that it does not adapt its weight as the series gets longer and longer. In the initial part of the series, the previous estimate is based on only a few prior observations and thus still has a fairly large forecasting error. In this

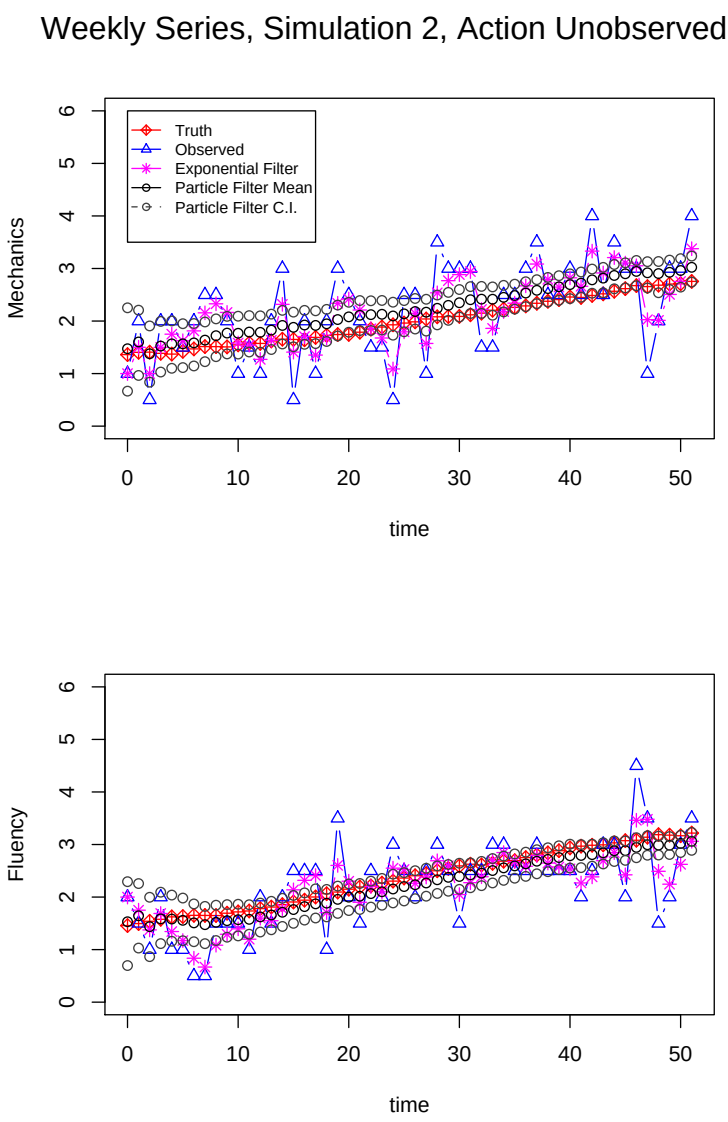

Figure **10 Time series estimates for Music Tutor example, action success unobserved, weekly meetings.**

case, the weight given to new observations should be fairly high. After several observations from the series, the forecasting error should decrease, and the previous estimate should be weighted more strongly in the mixture. The particle filter does this naturally (using the variance of the particle cloud as its estimate of the forecasting variance). The Kalman filter (Kalman & Bucy, 1961) also automatically adjusts the weights, and might produce a good approximation to the more expensive particle filter.

Another interesting feature of this simulation is the fact that the growth curves display periods of growth interspersed with flat plateaus. This shape is characteristic of a learning process that is a (latent) mixture of two growth curves. This seems to mimic behavior sometimes seen in real growth curves, where some students will show improvement at the beginning of a measurement period and some towards the end. If the SUCCESS variable were allowed to take on more than two values, the bowtie model could mimic situations with multiple growth curves at the cost of adding a few more parameters to the model.

These results do not take into account the uncertainty in the parameters due to estimation. The particle filter used the same parameters as the data generation process; thus, it was operating under nearly ideal conditions. It is still unknown how easy it will be to estimate parameters for this model. Fortunately, the problem can be divided into two pieces: estimating the parameters for the evidence models (for the benchmark tests) and estimating the parameters for the action models (for student growth). Evidence models can be estimated from cross-sectional data, for which it is relatively inexpensive to get large sample sizes. Estimating growth requires longitudinal data, which are always more expensive to collect.

The big advantage of the more complex model described in this paper is that it explicitly takes the instructor's actions from lesson to lesson into account when estimating the student's ability. Thus, the model provides a forecast of the student's ability under any educational *policy* (set of rules for determining an action), making it useful for instructional planning.

6 Planning

Although the cognitive processes of proficiency acquisition and deterioration are almost certainly more complex than the simple model described here, the model does capture some of their salient features. The model is hoped to be good enough for the purpose of filtering (improving the estimates of current student proficiency levels) and forecasting (predicting future proficiency levels). In particular, the instructor needs to be able to form a *plan*—a series of actions or strategies for selecting the actions at each lesson—that has the best probability of getting the student to the goal state.

The setup described here has most of the features of a Markov decision process. The previous sections show how the growth in proficiency can be modeled as a discrete time (or continuous time) Markov process. The evidence-centered design framework describes how observations can be made at various points in time. The last needed element is the specification of the utilities. First, assume that the costs associated with each test and each action are independent of time. Next, assume that there is a certain utility $u(\mathbf{S}_t, t)$ for the student being at state $\mathbf{S}_t$ at Time t, and let the total utility be $\sum_t u(\mathbf{S}_t, t)$. Then the utilities factor over time and all of the conditions of an MDP are met.

In general, the proficiency variables, $\mathbf{S}_t$, are unobserved. The action outcome X_t may be directly observable, indirectly observable (say through student self-report of practice time), or not observed at all. This puts us into a class of models called *partially observed Markov decision processes* (POMDPs), which require a great deal of computational resources to solve (Mundhenk, Lusena, Goldsmith, & Allender, 2000).

Another potential issue with this approach is the infinite time horizon. In theory, the total utility of a sequence of actions is the sum of the utilities across all time points. In order for that utility to be finite, either the time horizon must be finite (i.e., one semester), or future rewards must be discounted in relationship to current rewards. This is usually done by introducing a discount factor ϕ and setting the utility at Time t to be $\phi^t u(\mathbf{S}_t)$. This assumption may have interesting educational implications. For example, it is widely reported that students who acquire reading skills early have an advantage in many other subjects while students who acquire reading skills late have significant difficulties to

overcome.

The solution to a MDP is a *policy*—a mapping from the proficiency states of the student to actions (assignments given to the student). However, in the POMDP framework, as true proficiency levels are known, policies map from belief states about students to actions. If testing has a high cost in this framework, testing can be considered a different action (one that improves information rather than changes student state), and decisions about when to test can be included in the policy.

Finding solutions to POMDPs is hard (Mundhenk et al., 2000), but a number of approximate algorithms for solving POMDPs exist. The problem is still very difficult and heuristics for reducing the number of candidate actions are potentially useful. Here, reasoning about prerequisites can reduce the size of the search space. In particular, if the probability of success for an action is small, it can be eliminated. For example, any assignments that are much more difficult than the current best guess of the student's proficiency level can probably be safely pruned from the search space.

Suppose that the utility has a structure that gives a high reward if the student has reached a goal state, a certain minimal level of proficiency. This utility models much of the emphasis on standards in current educational practice. Then the POMDP model can be used to calculate the probability that the student will reach the goal—meet the standards—by a specified time point. In particular, the model can identify students for whom the probability of meeting the goal is small. These are *at-risk* students for whom the current educational policy is just not working. Additional resources from outside the system would need to be brought to bear on these students.

7 Discussion

The model described above captures many of the salient features of proficiency acquisition using only a small number of parameters. In particular, the number of parameters grows linearly in the number of prerequisites for each action, rather then exponentially. They are a compromise between producing a parsimonious model and a model faithful to the cognitive science. As such, this model may not satisfy either pure

statisticians or cognitive scientists, but it should have sufficient ability to forecast student ability to be useful, especially for the purposes of planning.

Even though the models are parsimonious, the ability to estimate the parameters from data is still largely untested. Markov chain Monte Carlo and other techniques may provide an approach to the problem, but weak identifiability of the parameters could cause convergence problems in such situations. These models need to be tested both through simulation experiments and applications to real data.

One part of the estimation problem is estimating the effects of a particular action. The literature on this topic is vast (von Davier & von Davier, 2005, provided a review). Usually both the pretest and posttest need to have high reliability, and often there is a problem of vertical scaling between the pretest and posttest.

One difficulty with estimating growth model parameters is experiments in classrooms rarely have pure randomized designs that easily allow causal attribution of observed effects. The goal of the model described in this paper is filtering (better information about current proficiency) and forecasting (predicting future state), not measuring change. It is sufficient to know that normal classroom activities plus a particular supplemental program have a given expected outcome. Untangling the causal factors has important implications for designing better actions but not immediate applicability to the filtering and forecasting problems. Even in situations where the goal is to make inferences about the effect of a particular program, the models developed here would be suitable as imputation models to fill in data missing due to random events (e.g., student absence on one of the test dates). As missing data can affect a large number of records in a study that observes students over a length of time, this capability represents a big improvement.

Many traditional models for student growth assume that growth for all students is along the same trajectory, with random errors about that common trajectory. Typical studies looking at proficiency trajectories (e.g., Ye, 2005) show a large number of different growth curves. The models described in this paper have only two growth curves (at each time point): one if the action succeeded and another if it did not (although a student can be on different curves in different time intervals). Whether this is adequate, or whether allowing

the SUCCESS variable to take on more than two states will produce an improvement, has yet to be determined. If the individual differences in rates are large, then either shortening the intervals between tests or increasing the reliability of the individual tests should diminish the impact of those differences in rate on the estimation of student proficiency.

Another potential problem is that in typical classroom assessments periodic benchmark assessments must be short in order to minimize the amount of time taken away from instruction. As test length plays a strong role in determining reliability, short benchmark tests will have modest reliability at best. In the Bayesian framework an estimate about a student's current ability is always a balance between the *prior* probability (based on the population and previous observations) and the *evidence* from the observations at the current time point. If the reliability of the benchmark tests is low, then the estimates of an examinee's current proficiency level will be based mostly on previous observations. Thus, the Bayesian framework already takes reliability into account.

There is a point of diminishing returns. If the test adds almost no information to the current picture of the student's proficiency, there is little point in testing. Likewise, there is little point in testing if the action will be the same no matter the outcome of the test. Actually, the lack of meaningful choices in the action space may prove to be the largest problem that this framework poses. No matter how good the diagnostic assessment system is, it will have little value if the instructor does not have a rich set of viable instructional strategies available to ensure that the educational goals are reached. On the other hand, a diagnostic assessment system that integrates explicit models of student growth together with a well-aligned set of instructional actions could add value for both students and instructors.

References

Black, P., & Wiliam, D. (1998a). Assessment and classroom learning. *Assessment in Education: Principles, Policy, and Practice*, *5*(1), 7–74.

Black, P., & Wiliam, D. (1998b). Inside the black box: Raising standards through classroom assessment. *Phi Delta Kappan*, *80*(2), 139-147.

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, *11*, 1-94.

Box, G. E. P., & Jenkins, G. M. (1976). *Time series analysis: Forecasting and control*. San Francisco, CA: Holden-Day.

Boyen, X., & Koller, D. (1998). Tractable inference for complex stochastic process. In R. Lopez de Mantaras & D. Poole (Eds.), *Proceedings of the fourteenth annual conference on uncertainty in artificial intelligence* (pp. 33–42). San Mateo, CA: Morgan Kaufmann.

Brockwell, P. J., & Davis, R. A. (2002). *Introduction to time series and forecasting* (2nd ed.). New York: Springer.

Dean, T., & Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computer Intelligence*, *5*, 142–150.

Dekhtyar, A., Goldsmith, J., Goldstein, B., Mathias, K. K., & Isenhour, C. (in press). Planning for success: The interdisciplinary approach to building bayesian models. *International Journal of Approximate Reasoning*.

Doucet, A., Freitas, N. de, & Gordon, N. (2001). *Sequential Monte Carlo methods in practice*. New York: Springer.

Eid, M. (2002). A closer look at the measurement of change: Integrating latent state-trait models into the general framework of latent mixed Markov modeling. *Methods of Psychological Research Online*, *17*(2). Retrieved October 30, 2007, from http://www.mpr-online.de/issue17/art3/eid.pdf.

Howard, R., & Matheson, J. (1981). Influence diagrams. In R. Howard & J. Matheson (Eds.), *Principles and applications of decision analysis*. Menlo Park, CA: Strategic

Decisions Group.

Junker, B., & Sijtsma, K. (2001). Cognitive assessment models with few assumptions, and connections with nonparametric item response theory. *Applied Psychological Measurement*, *25*, 258-272.

Kalman, R., & Bucy, R. (1961). New results in linear prediction and filtering theory. *Transactions of the ASME, Series D, Journal of Basic Engineering*, *83*, 95-107.

Koller, D., & Learner, U. (2001). Sequential Monte Carlo methods in practice. In A. Doucet, N. de Freitas, & N. Gordon (Eds.), *Sampling in factored dynamic systems* (pp. 445–464). New York: Springer.

Langeheine, R., & Pol, F. van de. (1990). A unifying framework for Markov modeling in discrete space and discrete time. *Sociological Methods and Research*, *18*, 416-441.

Liu, J. S. (2001). *Monte Carlo strategies in scientific computing.* New York: Springer.

Matheson, J. (1990). Using influence diagrams to value information and control. In R. M. Oliver & J. Q. Smith (Eds.), *Influence diagrams, belief nets and decision analysis.* New York: John Wiley & Sons.

Mathias, K. K., Isenhour, C., Dekhtyar, A., Goldsmith, J., & Goldstein, B. (2006). *Eliciting and combining influence diagrams: Tying many bowties together* (Technical Rep. No. TR 453-06). Lexington: University of Kentucky, Department of Computer Science.

Mislevy, R. J., Steinberg, L. S., & Almond, R. G. (2003). On the structure of educational assessment (with discussion). *Measurement: Interdisciplinary Research and Perspective*, *1*(1), 3-62.

Mundhenk, M., Lusena, C., Goldsmith, J., & Allender, E. (2000). The complexity of finite-horizon Markov decision process problems. *Journal of the ACM*, *47*(4), 681–720.

Murphy, K., & Russell, S. (2001). Rao-blackwellised particle filtering for dynamic Bayesian networks. In A. Doucet, N. de Freitas, & N. Gordon (Eds.), *Sequential Monte Carlo methods in practice* (pp. 499–515). New York: Springer.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference.* San Mateo, CA: Morgan Kaufmann.

R Development Core Team. (2005). *R: A language and environment for statistical computing*

[Computer software]. Vienna, Austria: R Foundation for Statistical Computing.

Raudenbush, S. (2001). Toward a coherent framework for comparing trajectories of individual change. In L. M. Collins & A. G. Sayer (Eds.), *New methods for the analysis of change* (pp. 35–64). Washington, DC: American Psychological Association.

Reye, J. (2004). Student modelling based on belief networks. *Internationl Journal of Artificial Intelligence in Education*, *14*, 63–96.

Rijmen, F., Vansteelandt, K., & De Boeck, P. (in press). Latent class models for diary method data: Parameter estimation by local computations. *Psychometrica*.

Ross, S. (1989). *Probability models* (4th ed.). New York: Academic Press.

Takikawa, M., D'Ambrosia, B., & Wright, E. (2002). Real-time inference with large-scale temporal bayes nets. In J. Breese & D. Koller (Eds.), *Proceedings of the 18th UAI conference.* San Mateo, CA: Morgan Kaufmann.

von Davier, M., & von Davier, A. A. (2005). *A micro-survey of models for individual change.* Unpublished manuscript, ETS, Princeton, NJ.

Vygotsky, L. (1978). *Mind in society: The development of higher mental processes.* Cambridge, MA: Harvard University Press.

Ye, F. (2005). *Diagnostic assessment of urban middle school student learning of pre-algebra patterns.* Unpublished doctoral dissertation, The Ohio State University.

Notes

[1] This example is loosely adapted from a metaphor that John Sabatini has used to explain his models for reading proficiency. I'm using this example rather than a more realistic one for two reasons. One, by creating an artificial example I can restrict the problem to a few dimensions, focusing on those aspects of the problem that require a new approach. Two, I expect that most of my audience have not recently been beginning readers, but that many of them have recent experiences with beginning musicians, either through their own interest or that of their children. Thus, I expect them to resonate better with the proposed models of cognition.

[2] Technically, there is one other restriction on the Markov decision process, which is the utility must also factor across time. This is discussed later.

[3] Alternatively, the ordered set of values `do`, `re`, `mi`, `fa`, `sol`, `la` could be used to make a true proficiency scale.

[4] Readers who are viewing a paper version of this report or for whom the embedded animation in not working properly can view the figure online at `http://www.ets.org/Media/Research/pdf/RR-07-40-Sime2m.mp4`.

[5] The way this color is rendered on various screens or printers may vary considerably, so interpret the gray values with some caution.

[6] This animation can be viewed at `http://www.ets.org/Media/Research/pdf/RR-07-40-Sime2mna.mp4`.

[7] This animation can be viewed at `http://www.ets.org/Media/Research/pdf/RR-07-40-Sime2wna.mp4`.

[8] Technically, this is a low-pass filter; high-pass filters for removing low frequency noise are also used.

Appendix A

Filters

In time series analysis, a *filter* is a function of a series of observations that produces a smoothed estimate of the trend of the series. The term comes from signal processing where the filter removes high frequency noise[8] from a signal. The filter provides a better estimate of the current position in the series. Usually the filter can be run for a few additional cycles to produce a forecast for the series.

This paper uses two different filters. Section A.1 describes the simple exponential filter. Section A.2 describes the more flexible and complex particle filter. The Kalman filter (Kalman & Bucy, 1961; see also Brockwell & Davis, 2002) lies somewhere between the two in complexity. If the growth process described by Equation 2 was a simple normal increment, then the Kalman filter would provide optimal predictions.

A.1 Exponential Filter

Assume interest in a time series $S_0, \ldots, S_t$, but a series $Y_0, \ldots, Y_t$ has been observed that reflects S_t with some added uncorrelated errors. A simple and often effective filter is formed by the weighted sum of Y_t and the previous estimate $\hat{S_{t-1}}$, as follows:

$$\hat{S}_t = \begin{cases} Y_0 & \text{for } t = 0 \\ \alpha Y_t + (1-\alpha)\hat{S_{t-1}} & \text{for } t > 0. \end{cases} \tag{A1}$$

This is called an *exponential filter* because when it is written as a sum of past observations, it looks like

$$\hat{S}_t = \sum_{k=0}^{t} \alpha(1-\alpha)^k Y_{t-k} \ .$$

Thus, past observations fade into the background at a rate $(1-\alpha)^k$. For large values of α, the filter output tracks the series Y_t fairly closely. For small values of α, the filter is much smoother. However, the summation form is truncated at Y_0. Thus for small α the initial estimate has a fairly high weight.

There doesn't seem to be a clear consensus about how to set an optimal value for the

filter parameter α. Brockwell and Davis (2002) described the filter but suggested plotting the filtered series for several values of α against the observed series and choosing the parameter based on the quality of the smoothed series.

Box and Jenkins (1976) noted that the filter is optimal for integrated moving average processes of order one, that is, a time series that can be represented by the equation:

$$Y_t = Y_{t-1} + e_t - \theta_1 e_{t-1} \ , \tag{A2}$$

where e_t is a white noise process with zero mean and constant variance, and where $\alpha = 1 - \theta_1$. However, estimating the coefficients of an integrated moving average process from a short series can be difficult. Another difficulty is that a value of α chosen as optimal for one student may not work well for another student.

Another issue is that the series described in Equation A2 when differenced has a zero expected value. Contrast this to Equation 2. If the tutor is applying a reasonable policy, the differenced should have a positive expected value. Accounting for this positive expectation requires an additional term for the expected effects of instruction in the filter equation. Without this term, the filter will slightly underestimate the true value. The smaller the value of α, the more these errors will accumulate to cause underestimates of the true series. However, adding an average drift parameter to the filter introduces a second parameter to estimate.

A.2 Particle Filter

When both the evidence model and the action model are based on normal distributions, the Kalman filter (Kalman & Bucy, 1961) provides an analytical method for calculating the posterior distribution over the student's proficiency at any time point. Unfortunately, the normality assumption falls down in two respects: (a) if the success variable is unobserved, then the process is a mixture of normals and not a single normal; and (b) the size of the drift depends on the distance between the unobserved true proficiency and the difficulty level of the chosen action. To get around those problems, numerical methods are looked at for filtering and forecasting.

The technique chosen below is the *particle filter* (Doucet et al., 2001; Liu, 2001), also known as sequential importance sampling. This is a Monte Carlo approximation, which estimates the posterior distribution through a randomly generated set of *particles*, $\mathbf{S}^{(1)}, \ldots, \mathbf{S}^{(N)}$. Each particle, $\mathbf{S}^{(n)} = \{\mathbf{S}_0^{(n)}, \mathbf{S}_1^{(n)}, \mathbf{S}_2^{(n)}, \ldots\}$, represents a possible trajectory of the student through proficiency space. At each time step, the particle filter assigns a weight, $w_t^{(n)}$, to each particle based on the likelihood of the observations sequence $\mathbf{Y} = \{\mathbf{Y}_0, \mathbf{Y}_1, \ldots, \mathbf{Y}_t\}$. This weight factors as follows:

$$w_t^{(n)} = \mathrm{P}(\mathbf{Y}_t | \mathbf{S}_t^{(n)}) w_{t-1}^{(n)} \ . \tag{A3}$$

The particle filter has a number of desirable properties. First, the accuracy is driven by the number of particles, N, and not the dimensionality of the proficiency space. Second, both the calculation of the trajectories and the weights factor in time and only require the forward data generation equations (as given in Section 5). Consequently, particle filters are mostly implemented by creating a "cloud" of particles at time point zero and moving them along a random trajectory at each time point, updating the weights along the way.

One disadvantage of the particle filter is that over time, the weights of most of the particles will approach zero. This means that the estimates will be based on only a few particles. To overcome this problem, the *bootstrap particle filter* periodically resamples particles from the current distribution, hopefully producing a new set of particles that spans a more meaningful part of the support of the distribution. The bootstrap filter takes a weighted sample using the current weight values and sets the weights of the newly sampled points to 1. Usually the bootstrap sampling is done according to a fixed time schedule (every so many time points), but other schemes are possible (Liu, 2001).

In estimating proficiencies in the music tutor problem, a series of *benchmark assessment* results, $\mathbf{Y}_0, \mathbf{Y}_1, \ldots$, and a series of *actions*, $\mathbf{a}_0, \mathbf{a}_1, \ldots$, are observed. There are two cases to consider. If the success of each action is known, then the data are augmented with the success indicator variables, $\mathbf{X}_0, \mathbf{X}_1, \ldots$. If not, then the particles are augmented with the success indicators and the success values must be simulated at each time step. Both variations of the music tutor example can be solved with the algorithm shown below.

1. *Simulate N particles, $\mathbf{S}_0^{(n)}$ from the initial population distribution for proficiency at Time 0.*

2. *Calculate the initial weights for these particles based on the likelihoods of the observables at the pretest benchmark, $\mathbf{Y}_0$.*

3. *For each time point, t:*

   (a) *If this is a bootstrap cycle (cycle number is divisible by the bootstrap resampling rate), then:*

      i. *Replace the current set of particles with a new set of particles randomly chosen according to a weighted sample using the current weights.*

      ii. *Set the current weight, $w_t^{(n)} = 1$.*

   (b) *If $\mathbf{X}_t$ is unknown, sample a value for $\mathbf{X}_t$ according to the distribution $\mathrm{P}(\mathbf{X}_t|\mathbf{S}_t^{(n)}, \mathbf{a}_t)$ (Equation 1).*

   (c) *Sample the value of the particle at the next time point by using the distribution $\mathrm{P}(\mathbf{S}_{t+1}|\mathbf{S}_t^{(n)}, \mathbf{a}_t, \mathbf{X}_t)$ (Equation 2).*

   (d) *Update the weights for the observations at time $t+1$ using Equation A3.*

Appendix B

Simulation Code

This section shows the code used for the simulation, written in the R (R Development Core Team, 2005) programming language. Section B.1 provides the functions used for the simulation, Section B.1 shows the implementation of the particle filter, and Section B.2 shows a script.

For those unfamiliar with R, it follows most of the usual conventions of algebraic computer languages (e.g., FORTRAN, C, Pascal, Java). However, a few features of R are worth noting:

- R is a functional language and functions can be passed as arguments to other functions. The `do.call` function is called to invoke the function. This allows both the simulator and particle filter to be written in abstract terms. Thus the line `est <- do.call(initial,list(nparticles))` in the particle filter creates the initial set of particles by calling the function `initial` with the argument `nparticles`; both the function and number of particles are passed as arguments to the particle filter itself.

- R treats matrixes and vectors as objects and implicitly loops over the elements. Thus, `A+B` is the elementwise addition of the matrixes `A` and `B`. The `apply()` and `sweep()` functions can be used to perform actions on rows and columns. For example, `apply(X,2,sum)` yields the column sums for the matrix `X`, and `sweep(X,1,Y,"/")` divides each row in the matrix `X` by the appropriate element of the vector `Y` (the length of `Y` is assumed to be the same as the number of rows of `X`).

B.1 Simulation Function

This section displays the code for setting up the simulation described in Section 5.1.

```
# These are files for the data generation routines.

## This function takes the proficiency expressed as a vector
## (Mechanics, Fluency) and generates a random lesson value.
evalLesson <- function(proficiency) {
```

```
  result <- proficiency %*% evalA +
    rnorm(length(evalSig))*evalSig
  result <- round(2*result)/2
  result <- ifelse((result < 0),0,result)
  result <- ifelse((result > 6),6,result)
  result
}

## Calculates likelihood of observation given proficiency.
## Assume Proficiency is a matrix with columns equal to the
## particles.  Assume observation is a vector (Mechanics,
## Fluency) Returns a vector of likelihoods, one for each
## particle
evalLike <- function (proficiency, observation) {
  if (!is.matrix(proficiency))
    proficiency <- matrix(proficiency,1)
  ## I'm ignoring the normalization constant and
  ## the rounding error
  ker <- sweep(sweep(proficiency %*% evalA,2,observation),
               2,evalSig,"/")
  exp(-apply(ker^2,1,sum)/2)
  ## exp(-sum( ( (proficiency %*% evalA -
  ## observation)/evalSig)^2)/2)
}

## Calculates success probability of action based
## on current proficiency level.
## Assumes action is a vectors of the form
## (Mechanics,Fluency)
## Assumes proficiency is a matrix with columns
## (Mechanics, Fluency) returns a vector of
## probabilities of success for the lesson.
actionSuccessP <- function (proficiency, action) {
  if (!is.matrix(proficiency)) {
    proficiency <- matrix(proficiency,1)
  }
  ##   delta <- threshlow <= action-proficiency &
  ##              action - proficiency <= threshhigh
  diff <- sweep(proficiency,2,action)
  delta <- sweep(diff,2,threshlow,">=") &
           sweep(diff,2,threshhigh,"<=")
  ##   q0 * prod(q1^(1-delta))
  ## Need to transpose matrix to get rows varying fastest
  qq1 <- apply(q1^(1-t(delta)),2,prod)
  outer(qq1,q0,"*")
}
```

```
## Calculates the increase rate based on current
## position and selected action.
## Assumes proficiency is a matrix and action is
## a vector
learnRate <- function(proficiency,action) {
  if (!is.matrix(proficiency)) {
    proficiency <- matrix(proficiency,1)
  }
#  eta - gamma*(proficiency-action)^2
  lam <- sweep(proficiency,2,action)^2
  sweep(-sweep(lam,2,gamma,"*"),2,eta,"+")
}

## This function calculates what happens between two lessons
## The the Action arguments are assumed to vectors of the
## form (Mechanics, Fluency).  The time argument should be a
## scalar.  Success argument is optionally observed.
## The proficiency and success arguments should be matrixes
## whose rows correspond to particles and have the
## (Mechanics, Fluency) pattern.
lessonEffect <-
function (proficiency, action, time=1,
          success = runif(length(proficiency)) <
                    actionSuccessP(proficiency,action)
          ){
  if (!is.matrix(proficiency)) {
    proficiency <- matrix(proficiency,1)
  }
  if (!is.matrix(success)) {
    success <- matrix(success, nrow(proficiency),
                      ncol(proficiency), byrow=TRUE)
  }
  rehearsal <- (rTimediff*success+rTimeconst)*time
  lambda <- learnRate(proficiency,action)
  sigmat <- sigma*sqrt(time)
  proficiency1 <- proficiency + lambda*rehearsal -
                  forgetRate*time
  proficiency1 <- proficiency1 +
    matrix(rnorm(length(proficiency))*sigmat,
           nrow(proficiency),ncol(proficiency), byrow=TRUE)
  proficiency1 <- ifelse (proficiency1 < 0, 0, proficiency1)
  proficiency1 <- ifelse (proficiency1 > 6, 0, proficiency1)
  list(success=success,proficiency=proficiency1)
}
```

```
## Generates the requested number of students
## returns a matrix with rows corresponding to students
## (particles) and columns corresponding to proficiencies.
randomStudent <- function(nstudents=1) {
  nvar <- length(popMean)
  z <- matrix(rnorm(nstudents*nvar),nstudents,nvar)
  result <- sweep(z%*%popA,2,popMean,"+")
  ##result <- as.vector(popA%*%rnorm(length(popMean))) +
  ## popMean
  result <- ifelse((result < 0),0,result)
  result <- ifelse((result > 6),6,result)
  colnames(result)<-names(popMean)
  result
}


## ncycles -- number of cycles to simulate
## estimator -- function used to estimate current student
## level from observables.
## policy -- function used to estimate next action from
## estimate
## evaluation -- function used to generate observables from
## current proficiencies
## advance -- function used to generate proficiencies at
## next time step from current time step.  Note:  assume
## that this returns a list of two values, success of action
## and new proficiencies
## initial -- function to generate random initial starting
## value for student.
## cycletime --- time between measurements (could be scaler
## or vector of length ncycles)
simulation <- function(ncycles,estimator,policy,
                       evaluation = evalLesson,
                       advance = lessonEffect,
                       initial = randomStudent,
                       cycletime=1) {
  proficiency <- do.call(initial,list())
  nvar <- length(proficiency)
  tnames <- paste("Time",0:ncycles)
  vnames <- colnames(proficiency)


  # We will fill these in as we go.
  truth <- matrix(NA,ncycles+1,nvar,
                  dimnames=list(tnames,vnames))
  obs <- truth
```

```
  est <- truth
  act <- truth
  suc <- truth
  if (length(cycletime) == 1) {
    cycletime <- rep(cycletime,ncycles)
  }
  ## Last cycle is stub only
  cycletime <- c(cycletime,0)

  # Now we start the simulation
  for (icycle in 1:ncycles) {
    truth[icycle,] <- proficiency
    observed <- do.call(evaluation,list(proficiency))
    obs[icycle,] <- observed
    estimate <- do.call(estimator,list(obs,est,act,suc))
    est[icycle,] <- estimate
    action <- do.call(policy,list(estimate))
    act[icycle,] <- action
    result <- do.call(advance,
                      list(proficiency, action,
                           time=cycletime[icycle]))
    suc[icycle,] <- result$success

    proficiency <- result$proficiency

  }
  truth[ncycles+1,]<-proficiency
  observed <- do.call(evaluation,list(proficiency))
  obs[ncycles+1,] <- observed
  sim <- data.frame(truth,obs,est,act,suc,cycletime)
  nsim <- paste("true",vnames,sep=".")
  nsim <- c(nsim,paste("obs",vnames,sep="."))
  nsim <- c(nsim,paste("est",vnames,sep="."))
  nsim <- c(nsim,paste("act",vnames,sep="."))
  nsim <- c(nsim,paste("suc",vnames,sep="."))
  nsim <- c(nsim,"cycletime")
  names(sim) <- nsim
  sim
}

expFilter <- function (obs,est,act,suc,
                       alpha = filterAlpha,
                       xi = meanInnovation) {
  nobs <- sum(!is.na(obs[,1]))
  if (nobs == 1) return(obs[1,])
  result <- alpha*obs[nobs,] + (1-alpha)*(est[nobs-1,]+xi)
```

```
  result <- ifelse((result < 0),0,result)
  result <- ifelse((result > 6),6,result)
  result
}

## Generates an action from a proficiency estimate
## Simple policy which tries to match precisely with
## estimate
floorPolicy <- function(estimate) {
  floor(2*estimate)/2
}
```

B.2 Particle Filter

This section displays the code for the particle filter (Section 7) as well as some graphics designed to work with the particle filter.

```
## Implementation of simple particle filter techniques.
## <resample> is number of time points between recycling,
## or zero to suppress resampling.

## In output list, success and map have entry of one size
## less than the est and weight lists (no entry for initial
## time point).

## <obs>, <act>, and <suc> are matrixes of observations,
## actions, and success values, where rows represent time
## points and columns represent proficiencies.  <suc>
## may be NULL, in which case random success values
## will be generated for each time point.

## <nparticles> controls the number of particles, and
## <resample> controls the number of cycles between
## bootstrap resampling of the particles.

## <cycletime> is the time length of a cycle
## (relative to the other rate parameters)

## Thes arguments should be functions (or function
## names) which calculate the given quantities.
## <obsLike> is the likelihood function for the
## benchmark test
## <successProb> is the probability of success for the
## chosen action
```

```
## <advance> is the forward step generation function of the
## stochastic process
## <initial> is the population prior distribution
bootFilter <- function (obs,act,suc=NULL,
                        resample=0, nparticles = 1000,
                        cycletime=1,
                        obsLike=evalLike,
                        successProb= actionSuccessP,
                        advance = lessonEffect,
                        initial = randomStudent) {
  obs <- as.matrix(obs)
  act <- as.matrix(act)
  if (!is.null(suc)) {
    suc <- as.matrix(suc)
  }
  ncycle <- nrow(obs)-1
  if (length(cycletime) == 1) {
    cycletime <- rep(cycletime,ncycle)
  }


  record <- list (est=list(ncycle), weight=list(ncycle),
                  map=list(ncycle-1), suc=list(ncycle-1))

  ##Initial setup
  ## The goal is to as much as possible use implicit
  ## looping functions for operations across particles
  ## and explicit looping functions for operations
  ## within a particle.
  est <- do.call(initial,list(nparticles))
  record$est[[1]] <- est
  weight <- do.call(obsLike,list(est,obs[1,]))
  record$weight[[1]] <- weight
  nvar <- ncol(est)

  ## Loop over time points
  for (icycle in 2:ncycle) {

    ## Resample if necessary
    if (resample >0 && icycle %% resample == 0) {
      ## Resample
      cat("Resampling␣at␣cycle",icycle,"\n")
      map <- wsample(nparticles,weight)
      record$map[[icycle-1]] <- map
      est <- est[map,]
      cat("Ave␣weight␣before␣=",mean(weight),
```

```
            "; after=", mean(weight[map]),"\n")
      weight <- rep(1,length(weight))
    } else {
      ## identity map
      map <- 1:nparticles
      record$map[[icycle-1]] <- map
    }

    ## Handle the success variable.  If it is observed, it
    ## provides evidence for the current proficiency.  If it
    ## is unobserved, we need to simulate it.

    sucP <- do.call(successProb,list(est,act[icycle,]))

    if (is.null(suc)) {
      ## Need to sample success values
      success <- matrix(runif(length(sucP)) <
                        sucP,nrow(sucP))
    } else {
      success <- matrix(suc[icycle,],
                        nparticles,nvar,byrow=TRUE)
      sv <- ifelse(success,sucP,1-sucP)
      weight <- weight * apply(sv,1,prod)
    }
    record$suc[[icycle-1]] <- success

    ## Advance position of particles
    est <- do.call(advance,
              list(est,act[icycle,],
                   success=success,
                   time=cycletime[icycle]))$proficiency
    record$est[[icycle]] <- est

    ## Evaluate new observations
    weight <- weight * do.call(obsLike,
                               list(est,obs[icycle,]))
    record$weight[[icycle]] <- weight

  }
  class(record) <- "ParticleFilter"
  record
}


## Returns a sample of indexes (1:length(weights)) given
```

```
## size using weights as weights
wsample <- function (size, weights) {
  w <- cumsum(weights)/sum(weights)
  ## Outer produces an array of true and false values,
  ## apply counts number of thresholds exceeded.
  apply(outer(runif(size),w,">"),1,sum)+1
}

## Calculates the mean of the particle filter at each time
## point. Argument should be a the output of a bootFilter
## routine. Output is a matrix whose rows correspond to time
## points and whose columns correspond to proficiencies
mean.ParticleFilter <- function (data) {
  vnames <- colnames(data$est[[1]])
  result <- matrix(NA,length(data$est),length(vnames))
  for (icycle in 1:length(data$est)) {
    result[icycle,] <-
      apply(data$est[[icycle]]*data$weight[[icycle]],2,sum) /
        sum(data$weight[[icycle]])
  }
  colnames(result) <- paste("pf",vnames,sep=".")
  result
}

## Calculates an upper and lower quantile of the particle
## distribution and each time point.
## <data> should be a particle filter output and
## <level> should be a vector of two numbers between zero
## and one for the upper and lower bound.
## Output is list of two matrixes whose rows correspond to
## time points and whose columns correspond to
## proficiencies, one for the upper bound, one for
## the lower bound.  This is not corrected for the number
## of particles.
ci.ParticleFilter <- function (data,level=c(.025,.975)) {
  vnames <- colnames(data$est[[1]])
  ub <- matrix(NA,length(data$est),length(vnames))
  lb <- matrix(NA,length(data$est),length(vnames))
  colnames(ub) <- colnames(lb) <- vnames
  for (icycle in 1:length(data$est)) {
    w <- data$weight[[icycle]]/sum(data$weight[[icycle]])
    for (var in vnames) {
      est <- data$est[[icycle]][,var]
      ord <- order(est)
      cuw <- cumsum(w[ord])
      os <- apply(outer(cuw,level,"<"),2,sum)
```

```
      os[2] <- os[2]+1  # Adjust to outer fences
      os <- ifelse(os<1,1,os)
      os <- ifelse(os>length(est),length(est),os)
      ci <- est[ord[os]]
      lb[icycle,var] <- ci[1]
      ub[icycle,var] <- ci[2]
    }
  }
  colnames(ub) <- paste("pfub",vnames,sep=".")
  colnames(lb) <- paste("pflb",vnames,sep=".")
  list(lb=lb,ub=ub)
}

## Produces a series of points for the particle filter, one
## at each time point.  This can be use to produce a series
## of static plots or an animation.
## <pf> is the output of a particle filter and
## <sim> is the simulation object used to gather it.
## If <delay> is non-zero, then a delay of so many seconds
## will be added between plots.  If <ask> is true,
## then R will prompt between each time frame.
## If <plotname> is non-null, then a series of png files
## will be generated with the given string as a prefix (it
## may contain a "/" to put the generated series in a
## subdirectory.  This image series can then be turned into
## a slide show or an animation. Haven't well thought
## out yet what happens if there are more than two
## proficiency dimensions.
plot.ParticleFilter <-
function(pf,sim,delay=0, plotname=NULL,
         ask=par("ask"), legend=FALSE,
         reftitle="Exponential␣Filter"
         ) {
  oldpar <- par(ask=ask)
  on.exit(par(oldpar))
  if (!is.null(plotname))
    fmt <- sprintf("%%s%%0%dd.png",
                   floor(log10(length(pf$est)))+1)
  for (i in 1:length(pf$est)) {
    if (!is.null(plotname))
      png(sprintf(fmt,plotname,i))
    g <- 1 - pf$weight[[i]]/max(pf$weight[[i]])
    gord <- order(-g)
    plot(pf$est[[i]][gord,],col=gray(g[gord]),
         main=paste("Time",i), pch=1,
         xlim=c(0,6),ylim=c(0,6))
```

```
    points(sim[i,1:2],col="red",pch=9,cex=2)
    points(sim[i,3:4],col="blue",pch=2,cex=2)
    points(sim[i,5:6],col="magenta",pch=8,cex=2)
    if (any(legend !=FALSE)) {
      legend(legend,legend=c("True Proficiency",
                      "Benchmark Test",
                      reftitle, "Particle",
                      "(Colored according", "to weight)"),
       col=c("red","blue","magenta","black",
             grey(.5),grey(.75)),
       pch=c(9,2,8,1,1,1), pt.cex=2)

    }
    if (!is.null(plotname)) dev.off()
    else if (delay >0) Sys.sleep(delay)
  }

}

## This produces a series of time series plots, one for each
## proficiency variable.
## The <simb> input is a data frame formed by joining the
## simulator output, with the mean.ParticleFilter and
## ci.ParticleFilter output.  Currently makes lots of
## assumptions based on the Music Example (hardwired
## column numbers and names).
## If <pfonly> is TRUE, then suppressed printing of
## <observed> and <estimate> columns from simulator.
## Need better pass through of graphics parameters
## to adjust for differences between X11 and PDF
## graphics
plot.simboot <-
function (simb,pfonly=FALSE,reftitle="Running Ave") {
  time <- (1:nrow(simb))-1

  plot(time,simb[,1],type="b",pch=9,col="red",
       ylim=c(0,6),ylab="Mechanics")
  if (!pfonly) {
    points(time,simb[,3],type="b",pch=2,col="blue")
    points(time,simb[,5],type="b",pch=8,col="magenta")
  }
  points(time,simb[,12],type="b",pch=1,col="black")
  points(time,simb[,14],type="b",pch=1,lty=2,
         col=gray(.25))
  points(time,simb[,16],type="b",pch=1,lty=2,
         col=gray(.25))
```

```
  if (pfonly)
    legend(c(0,max(time)/3),c(6,4.5),
           c("Truth", "Particle␣Filter␣Mean",
             "Particle␣Filter␣C.I."),
           col=c("red","black",gray(.25)),
           pch=c(9,1,1),cex=.75,
           lty=c(1,1,2))
  else
    legend(c(0,max(time)/3),c(6,3.5),
           c("Truth","Observed",reftitle,
             "Particle␣Filter␣Mean",
             "Particle␣Filter␣C.I."),
           col=c("red","blue","magenta","black",gray(.25)),
           pch=c(9,2,8,1,1),cex=.75,
           lty=c(1,1,1,1,2))


  plot(time,simb[,2],type="b",pch=9,col="red",
       ylim=c(0,6),ylab="Fluency")
  if (!pfonly) {
    points(time,simb[,4],type="b",pch=2,col="blue")
    points(time,simb[,6],type="b",pch=8,col="magenta")
  }
  points(time,simb[,13],type="b",pch=1,col="black")
  points(time,simb[,15],type="b",pch=1,lty=2,col=gray(.25))
  points(time,simb[,17],type="b",pch=1,lty=2,col=gray(.25))
}
```

B.3 Sample Script

This section displays the script used to generate the examples. Simulation 2 in the code corresponds to Section 5.2.

```
#This is the actual Simulations

## This function collects all of the initial simulation
## parameters in one place to make it easy to re-run
## simulations.
resetParametersSimulation2 <- function () {
  ## Regression coefficients for multivariate regression.
  evalA <<- matrix(c(.7,.3,.3,.7),2,2)
  ## Noise std for multivariate regression.
  evalSig <<- c(.65,.65)
```

```
  ## Constant noise term in noisy-or model
  q0 <<- c(Mechanics=.8,Fluency=.9)
  ## Upper and lower bounds for delta in noisy-or model
  threshlow <<- c(Mechanics=-1,Fluency=-2)
  threshhigh <<- c(Mechanics=2,Fluency=1)
  ## skill bypass parameters in noisy-or model
  q1 <<- c(Mechanics=.4,Fluency=.5)

  ## Constant term in learning rate
  eta <<- c(Mechanics=.2,Fluency=.2)
  ## Zone of Proximal Development term in learning rate
  gamma <<- c(Mechanics=.1,Fluency=.15)

  ## Constant parameters of the learning process
  forgetRate <<- c(Mechanics=.02,Fluency=.01)

  ## R(t) = (rTimediff*success + rTimeconst)*t
  rTimediff <<- .9
  rTimeconst <<- .1

  ## Variance of learning processes is sigma^2*t
  sigma <<- c(Mechanics=.02,Fluency=.02)

  ## Mean and variance of the population.
  ## popA %*% t(popA) is the variance
  popA <<- matrix(c(.6,.4,.4,.6),2,2)
  popMean <<- c(Mechanics=1.5,Fluency=1.5)

  timeStep <<- 1
  varInnovation <<- sigma*sigma*timeStep +
    (timeStep*rTimediff)^2*q0*(1-q0)*eta*eta
  filterAlpha <<- .5
  meanInnovation <<- 0

}

##########################################################
## Simulation 2, lower success rate
## This is the one used in Section 8.3 of the paper.

resetParametersSimulation2()


sime2w <- simulation(52,expFilter,floorPolicy,
                     cycletime=.25)
```

```
boote2w <- bootFilter(sime2w[,3:4],sime2w[,7:8],
                      sime2w[,9:10],
                      cycletime=sime2w[,11],
                      nparticles=1000,resample=4)

## These two plots produce "animations" of the series
par(mfrow=c(1,1))
plot.ParticleFilter(boote2w,sime2w,delay=.25,
                    legend="topleft")

plot.ParticleFilter(boote2w,sime2w,
                    plotname="Sime2w/Music",
                    legend="topleft")

## Calculate confidince bands
ci2wb2 <- ci.ParticleFilter(boote2w)
mean2wb2 <- mean.ParticleFilter(boote2w)
sime2wb2 <- data.frame(sime2w[1:52,],mean2wb2,
                      ci2wb2$lb,ci2wb2$ub)

## Count the number of times the true series passes
## outside of the confidence band.
sum(sime2wb2$true.Mechanics < sime2wb2$pflb.Mechanics ||
    sime2wb2$true.Mechanics > sime2wb2$pfub.Mechanics)
sum(sime2wb2$true.Fluency < sime2wb2$pflb.Fluency ||
    sime2wb2$true.Fluency > sime2wb2$pfub.Fluency)


pdf("Sime2wB2TS.pdf",width=6,height=9)
par(mfrow=c(2,1),oma=c(0,0,0,0))
plot.simboot(sime2wb2, reftitle="Exponential␣Filter")
mtext("Weekly␣Series,␣Simulation␣2,␣Action␣Observed",
      outer=TRUE,cex=1.5,line=-2)
dev.off()
system("open␣Sime2wB2TS.pdf")


##########################################################
## Sim2 conditions, monthly series

resetParametersSimulation2()

sime2m <- simulation(12,expFilter,floorPolicy,
                     cycletime=1)
```

```
boote2m <- bootFilter(sime2m[,3:4],sime2m[,7:8],
                      sime2m[,9:10],
                      cycletime=sime2m[,11],
                      nparticles=1000,resample=4)

pdf("Sime2mBootstrap.pdf",width=6,height=6)
par(mfrow=c(3,4),oma=c(0,0,3,0))
plot.ParticleFilter(boote2m,sime2m)
mtext("Monthly␣Series,␣Action␣Observed,Bootstrap␣Filter",
      outer=TRUE,cex=1.5,line=0)
dev.off()
system("open␣Sime2mBootstrap.pdf")

## Animations of estimation
plot.ParticleFilter(boote2m,sime2m,
                    plotname="Sime2m/Music",
                    legend="topleft")
plot.ParticleFilter(boote2m,sime2m,delay=.25,
                    legend="topleft")

ci2mb2 <- ci.ParticleFilter(boote2m)
mean2mb2 <- mean.ParticleFilter(boote2m)
sime2mb2 <- data.frame(sime2m[1:12,],mean2mb2,
                      ci2mb2$lb,ci2mb2$ub)

## Number of times true series falls outside of
## confidence bands
sum(sime2mb2$true.Mechanics < sime2mb2$pflb.Mechanics ||
    sime2mb2$true.Mechanics > sime2mb2$pfub.Mechanics)
sum(sime2mb2$true.Fluency < sime2mb2$pflb.Fluency ||
    sime2mb2$true.Fluency > sime2mb2$pfub.Fluency)


pdf("Sime2mB2TS.pdf",width=6,height=9)
par(mfrow=c(2,1),oma=c(0,0,0,0))
plot.simboot(sime2mb2,reftitle="Exponential␣Filter")
mtext("Monthly␣Series,␣Action␣Observed",
      outer=TRUE,cex=1.5,line=-2)
dev.off()
system("open␣Sime2mB2TS.pdf")


## Reported table of results
xtable(as.matrix(sime2m[,c(1:4,7:10)]),
       digits=c(2,2,2,1,1,1,1,0,0))
```

```
#########################################################
## Simulation 2, action unobserved

boote2wna <- bootFilter(sime2w[,3:4],sime2w[,7:8],
                        cycletime=sime2w[,11],
                        nparticles=1000,resample=4)

## Animations
par(mfrow=c(1,1))
plot.ParticleFilter(boote2wna,sime2w,delay=.5,
                    legend="topleft")
plot.ParticleFilter(boote2wna,sime2w,
                    plotname="Sime2wna/Music",
                    legend="topleft")

ci2wb2na <- ci.ParticleFilter(boote2wna)
mean2wb2na <- mean.ParticleFilter(boote2wna)
sime2wb2na <- data.frame(sime2w[1:52,],mean2wb2na,
                         ci2wb2na$lb,ci2wb2na$ub)

## Count the number of times the true series passes
## outside of the confidence band.
sum(sime2wb2na$true.Mechanics < sime2wb2na$pflb.Mechanics ||
    sime2wb2na$true.Mechanics > sime2wb2na$pfub.Mechanics)
sum(sime2wb2na$true.Fluency < sime2wb2na$pflb.Fluency ||
    sime2wb2na$true.Fluency > sime2wb2na$pfub.Fluency)

pdf("Sime2wB2NATS.pdf",width=6,height=9)
par(mfrow=c(2,1),oma=c(0,0,0,0))
plot.simboot(sime2wb2na,reftitle="Exponential␣Filter")
mtext("Weekly␣Series,␣Simulation␣2,␣Action␣Unobserved",
      outer=TRUE,cex=1.5,line=-2)
dev.off()
system("open␣Sime2wB2NATS.pdf")



#########################################################
## Now on monthly scale

boote2mna <- bootFilter(sime2m[,3:4],sime2m[,7:8],
                        cycletime=sime2m[,11],
                        nparticles=1000,resample=4)

pdf("Sime2mnaBoote.pdf",width=6,height=6)
```

```
par(mfrow=c(3,4),oma=c(0,0,3,0))
plot.ParticleFilter(boote2mna,sime2m)
mtext("Monthly␣Series,␣Action␣Unobserved,Bootstrap␣Filter",
      outer=TRUE,cex=1.5,line=0)
dev.off()
system("open␣Sime2mnaBoote.pdf")

## Animations
plot.ParticleFilter(boote2mna,sime2m,
                    plotname="Sime2mna/Music",
                    legend="topleft")
plot.ParticleFilter(boote2mna,sime2m,delay=.5,
                    legend="topleft")

ci2mnab2 <- ci.ParticleFilter(boote2mna)
mean2mnab2 <- mean.ParticleFilter(boote2mna)
sime2mnab2 <- data.frame(sime2m[1:12,],mean2mnab2,
                         ci2mnab2$lb,ci2mnab2$ub)

## Number of times true series falls outside of
## confidence bands
sum(sime2mb2$true.Mechanics < sime2mb2$pflb.Mechanics ||
    sime2mb2$true.Mechanics > sime2mb2$pfub.Mechanics)
sum(sime2mb2$true.Fluency < sime2mb2$pflb.Fluency ||
    sime2mb2$true.Fluency > sime2mb2$pfub.Fluency)


pdf("Sime2mnaB2TS.pdf",width=6,height=9)
par(mfrow=c(2,1),oma=c(0,0,0,0))
plot.simboot(sime2mnab2,reftitle="Exponential␣Filter")
mtext("Monthly␣Series,␣Action␣Unobserved",
      outer=TRUE,cex=1.5,line=-2)
dev.off()
system("open␣Sime2mnaB2TS.pdf")
```