Online Embedded Assessment for Dragoon, Intelligent Tutoring System

by

Sachin Grover

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2015 by the
Graduate Supervisory Committee:

Kurt VanLehn, Chair
Erin Walker
Ihan Hsiao

ARIZONA STATE UNIVERSITY

December 2015

ABSTRACT

Embedded assessment constantly updates a model of the student as the student works on instructional tasks. Accurate embedded assessment allows students, instructors and instructional systems to make informed decisions without requiring the student to stop instruction and take a test. This thesis describes the development and comparison of several student models for Dragoon, an intelligent tutoring system. All the models were instances of Bayesian Knowledge Tracing, a standard method. Several methods of parameterization and calibration were explored using two recently developed toolkits, FAST and BNT-SM that replaces constant-valued parameters with logistic regressions. The evaluation was done by calculating the fit of the models to data from human subjects and by assessing the accuracy of their assessment of simulated students. The student models created using node properties as subskills were superior to coarse-grained, skill-only models. Adding this extra level of representation to emission parameters was superior to adding it to transmission parameters. Adding difficulty parameters did not improve fit, contrary to standard practice in psychometrics.

DEDICATION

In memory of my Father

Darshan Lal Grover

(1948 - 2014)

ACKNOWLEDGMENTS

I would like to express my most sincere thanks and gratitude to my advisor Dr. Kurt VanLehn, who has guided me throughout my graduate study and gave me the freedom and support to conduct this research. Working with him has been one of the most enriching experience both professionally and personally. I am also grateful to my thesis committee members, Dr. Erin Walker and Dr. Ihan Hsiao for their valuable time serving on my thesis committee.

I also want to thank Dr. Brett Van de Sande for his time to review my thesis as that actually helped me understand the improvements that I needed. His continuous mentoring has helped me analyse my work from different perspective. I am also thankful to Dr. Jon Wetzel for his support, that had given me freedom to express my ideas through my thesis.

I thankful to my team members in Dragoon lab Mervyn, Bahar, Tushar, Ritesh, Pradeep, Vallabh and Avaneesh for their constant support and motivation. I also want to thank my friends from FACT lab Kenil, Mridul, Ayush, Jasdeep and Adithya for their support during the last year of my thesis.

I am thankful to all the students that came for the study for Dragoon, especially Rohit and Smriti for letting me do sanity testing for it. I am thankful to my roommates Abinash, Shatadal, Shashi, Aniket, Anurag, Yash and Ritesh for putting up with me and making sure that I did my work on time.

I am thankful to my friends Arun and Yash for their presence and guidance throughout my thesis. All the discussions with them over my thesis, had helped to give it a better shape.

I am also thankful to all my friends from Ramjas, Soma, Pallavi, Ankit, from NIT Rourkela, Baltej, Ravi, Vinay, Shrinivas, Atul, Ganesh, Anupam, from Sapient, Simar, Yash, Surjeet, Rashik, Anish and Sahiba and also from ASU, Sowmya, Vandita, as their friendship has made me who I am today. I am also thankful to Anuj and Kushal for their brotherly support and guidance.

I am grateful for the support from my elder brothers, Naveen and Kapil. Without their belief in me, my higher education wouldn't have been possible. I want to thank my mother and grand mother Prabha Grover and Raj Rani Grover for their blessings. I also want to thank my father Late Darshan Lal Grover, as I miss him dearly and work harder to fulfil his wishes. I feel blessed by the presence of Smriti Sahu, as her constant support pushes me to be a better student everyday.

TABLE OF CONTENTS

Page

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTER

1 INTRODUCTION 1

2 DRAGOON: INTELLIGENT TUTORING SYSTEM 2

2.1 Interface and Notation 2

2.2 Problems in Dragoon 6

2.3 Feedback and Hints 6

2.4 Knowledge Components 7

3 GRAPHICAL MODELS AND STUDENT ASSESSMENT 11

3.1 Dynamic Bayesian Networks (DBN) 11

3.2 Inference in Graphical Models 13

3.3 Calibrating DBNs 14

3.4 Bayesian Knowledge Tracing 15

3.5 Feature Aware Student Knowledge Tracing (FAST) 16

3.6 BNT-SM 18

4 STUDENT MODELING METHODS FOR DRAGOON 19

4.1 Subskills 20

4.2 Difficulty 22

4.3 Adding Features to Emission vs. Transition Probabilities 23

4.4 Implementation Detail 24

4.5 A BKT Model with Subjective Probabilities 26

5 RESULTS AND DISCUSSION 30

5.1 Data collection 30

5.2 Results using BNT-SM 33
5.3 Results using FAST 35
5.4 Comparison of Transition and Emission Subskill Models 40
5.5 Comparing Schemas 43
5.6 Evaluation of the Addition of Difficulty Parameters 44
5.7 Summary 46
6 EVALUATION WITH SIMULATED STUDENTS 47
6.1 Defining the Simulated Students 47
6.2 Reliability 49
6.3 Validity 50
7 CONCLUSION AND FUTURE WORK 52
REFERENCES 57

LIST OF TABLES

Table Page

4.1 Skill and Subskill Combination Tried for Model Fitting in Dragoon. The Numbers in the Bracket Show the Count of Skills and Subskills in the Data. "Control" Model Was Also Part of the Results Group. 22

5.1 Schema Counts per Student While Interacting with Dragoon. 32

5.2 Total Count of Schema Application Opportunity Distribution for Correct/incorrect Student Response. 32

5.3 Number of Nodes and Schemas in the Problems Used for the Study. . . . 33

5.4 AUC Results for BNT-SM. 34

5.5 AUC Results for FAST Algorithm with Subskills as Emission Constant Features 36

5.6 AUC Results for Fast Algorithm with Subskills as Emission Constant Features 38

5.7 AUC Results for Fast Algorithm with Subskills and Difficulty Parameters as Emission Features. 39

5.8 Parameter Values from Fast for a Skill in Transition Features Subskill Model. 42

5.9 Parameter Values from Fast for a Skill in Emission Features Subskill Model. 42

5.10 Parameter Values for Skills in Group6. 43

6.1 Final Probability of Mastery for Exponential Schemas Using Assessment Model, with Transition Constant Features. 49

LIST OF FIGURES

Figure Page

2.1 Completed Model for Population Problem in Immediate Feedback Mode 4

2.2 Node Editor in Dragoon 5

2.3 Model Construction Through Problems in Dragoon 6

2.4 Schema Overlap in Dragoon in Node *Bus Fleet* 9

2.5 Interface for Author to Define Schemas with Difficulty Parameters 10

3.1 Dynamic Bayes Network with X Hidden Variables and E Evidence Variables 12

3.2 A DBN for a Skill Used in Student Modeling Using Plate Notation 13

4.1 DBN with Difficulty Parameters for Sub-Skills 23

5.1 ROC Curve using BNT SM 34

5.2 AUC Results for FAST Algorithm for Emission Constant Model for Skill Defined as Schemas_nodeType. 37

5.3 AUC Results for FAST Algorithm for Emission Constant Model for Skill Defined as Schemas. 37

5.4 AUC Results for FAST Algorithm for Transition Constant Model for Skill Defined as Schemas_nodeType. 38

5.5 AUC Results for FAST Algorithm for Transition Constant Model for Skill Defined as Schemas. 39

5.6 ROC Curve Using FAST for Difficulty Parameters and Subskills as Transition Features. 40

Chapter 1

INTRODUCTION

This thesis addresses the problem of creating a student model (embedded assessment) for Dragoon, which is an intelligent tutoring system for teaching students how to construct models of dynamic systems. The assessment will be used to recommend problems to students. However, the choice of a recommendation policy is not part of this thesis and, for testing purposes, problems were presented to the student randomly.

Several student modeling methods were developed and compared. They were evaluated in two ways. First, the fits of the model to human data was measured. Second, the methods assessed synthetic students, and their accuracy was measured. See the table of contents for a brief overview of the thesis chapters.

Chapter 2

DRAGOON: INTELLIGENT TUTORING SYSTEM

Dragoon is an example-tracing tutor (Koedinger et al., 2004) which means that it compares a student step to steps authored by a human expert, and uses this comparison to provide feedback to the student. Thus, Dragoon's inner loop (VanLehn, 2006) is rather simple and will not be described here. Dragoon currently has no outer loop, which means that it can only coach a student through solving a problem and cannot recommend or select a problem for the student to do next. In order to do adaptive task selection or recommendation, it needs to have a student model. Thus, this thesis is on the critical path for improving Dragoon.

Dragoon teaches students how to create models of dynamic systems. A *system* is a part of the real world, and a *dynamic system* is a system which changes with time. A record of the system's change as a function of time is called its *behavior*. A *model* is an expression in a modeling language that can be interpreted by a computer. Executing a model generates its behavior. An accurate model will generate predictions that match the target system's actual behavior VanLehn et al. (forthcoming).

This chapter explains Dragoon in detail. First, it talks about the interface and the notation. Then it presents the step analyzer as well as hints and feedback given to the students. The third section explains the knowledge components in detail.

2.1 Interface and Notation

Dragoon is a step-based tutoring system. In order to construct a model, students construct *nodes*, and in order to construct a node, students enter a *step* into a node editor. Students get feedback on each step, and after several failed attempts at a step, Dragoon shows the person how to do the step by doing the step itself. Thus, the raw input to the student modeling component is a series of steps attempted by the student, each annotated with whether the student got the step right on the first try.

To understand this process in detail, let's take an example problem. The following is a system description that is given to the student:

In the year 2013, the population in the city of Phoenix was 1.51 million. Every year the population grew by 1.67%. Create a model for the population of the city and see how much it has increased by the year 2050, assuming that growth rate remains constant in that time.

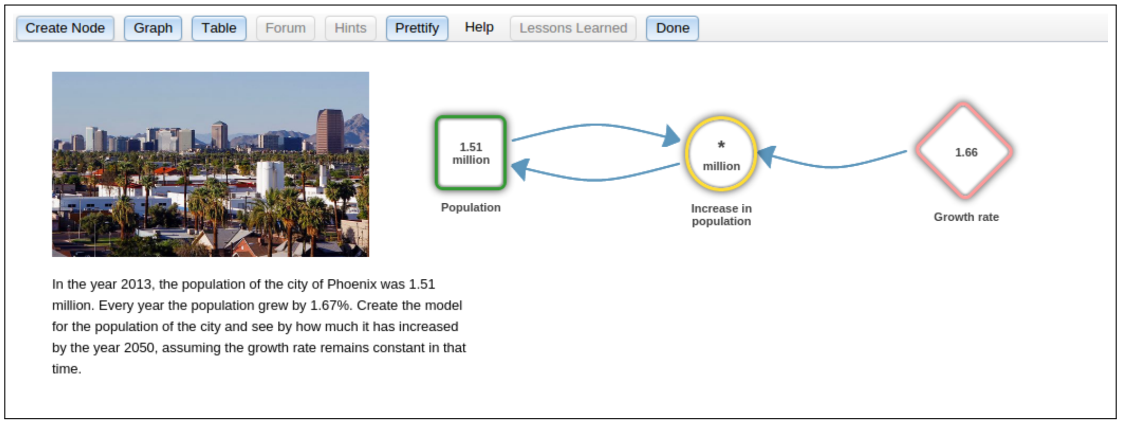
The students are asked to create a model of this system using Dragoon. Figure 2.1 shows the screen after a model has been constructed.

In Dragoon, the quantities that comprise the model are represented as *Nodes.* Nodes are of three types:

1. *Parameter* - is a constant variable. For example, as the population growth rate does not change throughout the time period, so it will be a parameter in the model. In figure 2.1, diamond signifies a parameter.

2. *Function* - corresponds to a mathematical relationship between nodes, like net growth in population = Population * Growth Rate. In figure 2.1, circle signifies a function node.

3. *Accumulator* - as the name suggests, this is where the value of a quantity gets accumulated. After each time slice the value of the node updates based on its value in the previous time slice and the relationship with other quantities. Like in the presented problem, new Population = old Population + net growth in Population. In figure 2.1, square signifies an accumulator.



In the year 2013, the population of the city of Phoenix was 1.51 million. Every year the population grew by 1.67%. Create the model for the population of the city and see by how much it has increased by the year 2050, assuming the growth rate remains constant in that time.

Figure 2.1: Completed Model for Population Problem in Immediate Feedback Mode

A node has five properties which are used to define the quantity that it represents. Values are set in the node editor by either selecting the correct value from a dropdown menu or by entering it through the keyboard. Setting a property corresponds to a step. The properties are:

1. Description - name of the node
2. Type - node type as described above.
3. Initial value - The value of the node at the start of time. Only for parameter and accumulator nodes.
4. Units - units of the quantity represented by the node.
5. Equation - relation with which the node is related to other nodes in the model. Only for function and accumulator nodes.

Figure 2.2 shows the node editor used in Dragoon. All the properties that the student needs to fill are shown with question marks which provide help to student to fill the node completely explaining what these properties are.

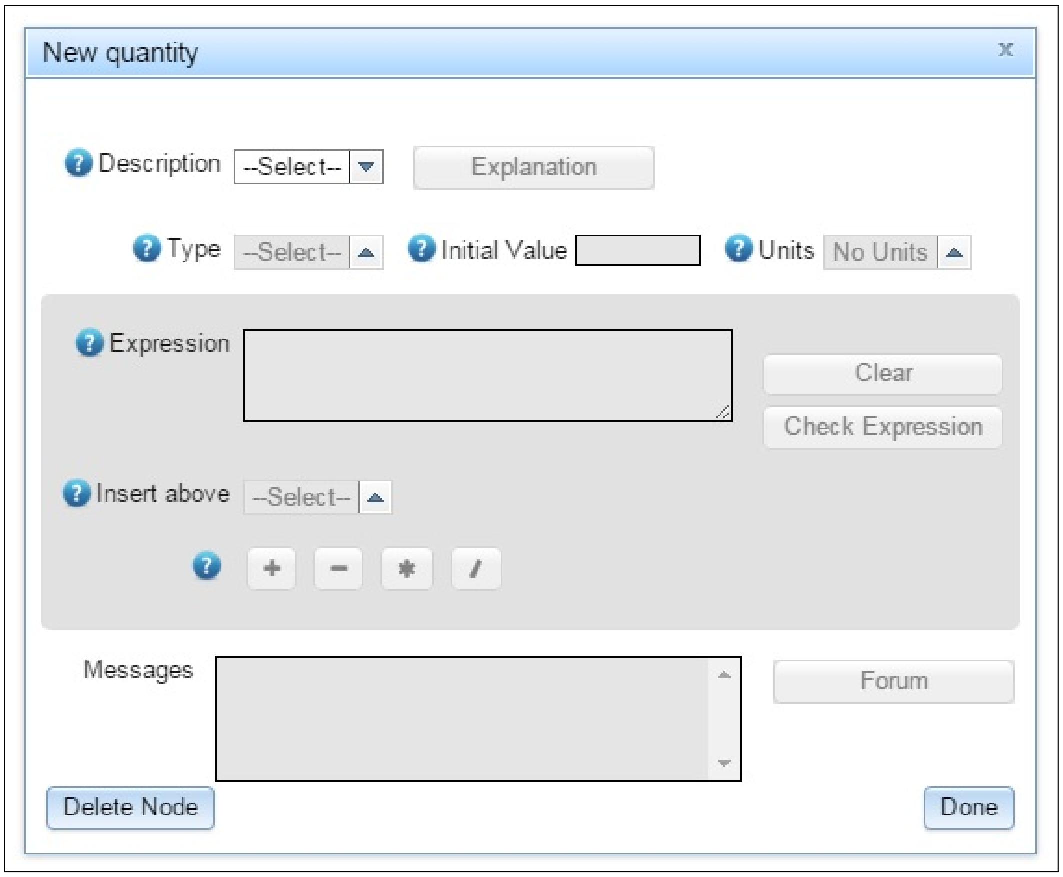


Figure 2.2: Node Editor in Dragoon

On the user interface, arrow-like links show the inputs to function and accumulator nodes. After completing the model, a student can click on the “Graph” and “Table” buttons to see the model behavior. For the above problem, they would show an increase in population as a function of time, for the years 2013 to 2050. Figure 2.1 shows the completed model. It also shows the interface of Dragoon in which we present the problem on the right and on the left student creates his model. At the top there are buttons like Graph and Table to check the behavior of the model student has constructed.

2.2 Problems in Dragoon

Figure 2.1 also shows the interface of Dragoon. Any problem in Dragoon requires a student to construct nodes using node editor shown in figure 2.1. Student completes all the node properties to completely define a node, which are joined together to form schemas which will be explained in section 2.4. These join together to form a complete problem in Dragoon. Figure 2.3 shows the complete flow for a problem in Dragoon while constructing a model.

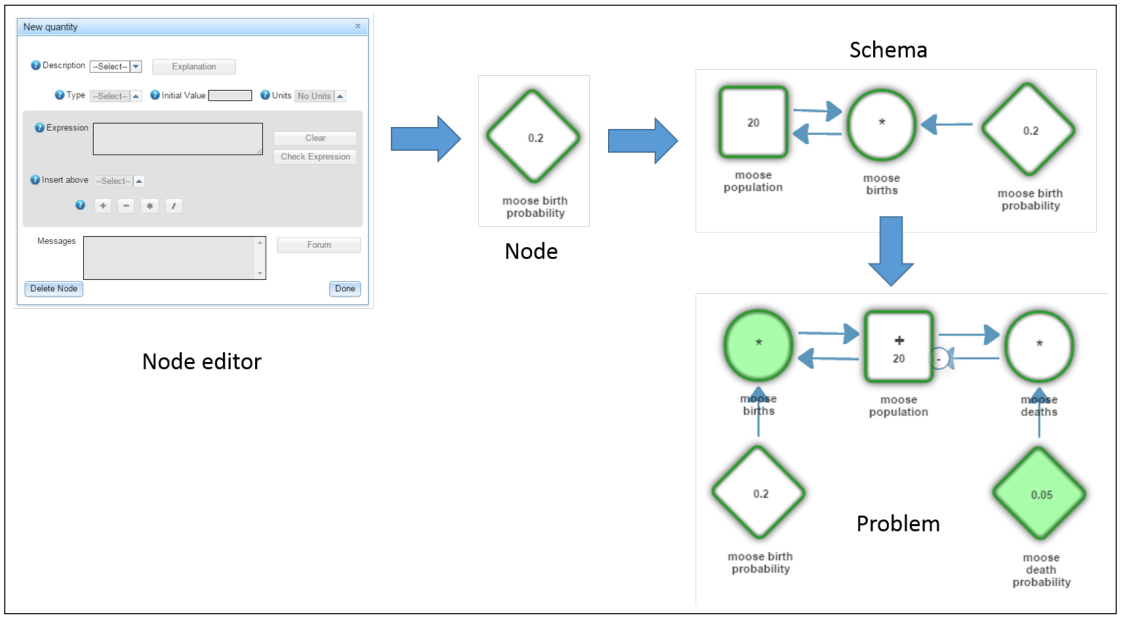


Figure 2.3: Model Construction Through Problems in Dragoon

2.3 Feedback and Hints

Feedback and hints in Dragoon depend on the mode in which the student solves a problem. There are three modes available -

- *Immediate feedback* - gives minimal feedback to the student after each step. The feedback is given by showing green color for correct solution, red color for a wrong solution and yellow color if the answer is given by Dragoon.

- *Delayed feedback* - does not give any feedback after each step. When a student opens the graph and table values, then they are told whether their model matches the author solution or not.
- *No feedback* - does not give any feedback.

There is no hint button in Dragoon, but it does provide the correct entry for a step (a “bottom out” hint) if the student has entered several incorrect attempts. This hint is provided only in the immediate feedback mode.

2.4 Knowledge Components

Constructing a Dragoon model when given a succinct description of a system is nearly the same as solving a mathematical word problem, that is, constructing an equation or set of equations when given a succinct description of the system. The knowledge required for solving word problems can be represented as schemas, where each schema pairs something that matches the system description with something that output an equation or set of equations. Marshall et al. (1989), found that single-step arithmetic problems could be divided into 5 fundamental schemas: Change, Group, Compare, Restate, and Vary. Mayer (1981), found that the algebra word problems in 10 textbooks could be covered by about 90 schemas. For instance, there were 10 schemas for river crossing problems, 5 schemas for interest problems and so forth. A complex arithmetic or algebra word problem may require applying two or more schemas.

Dragoon assumes that its students' knowledge can be represented as schemas. Each schema has a part that matches the system description as a template for a model. The template consists of a set of nodes, but the initial values, names, descriptions and units have not been filled in. The types and equations have been filled in.

To apply a mentally held schema, the student notices that it can be applied, then enters the appropriate nodes into Dragoon, filling in the missing information (names, descriptions, values and units) as the nodes are constructed. For example, the model shown in figure 2.1 is an instance of the exponential growth. The only difference between the schema and the model of figure 2.1 is that the model has specific information from the problem entered into the description, value, units and equation properties of the nodes.

To represent a particular student's competence at constructing Dragoon models, this thesis assumes that every schema has a probability of mastery associated with it. If a student has mastered a schema, then the student will apply the schema correctly almost every time it is needed. The probability of making a mistake when apply a mastered schema is typically low, e.g., 0.1, and called the slip probability. Different schemas can have different slip probabilities.

Following Mayer (1981), Dragoon, schemas are grouped under schema families, such as basic electronics, population ecology and kinematics. There is one such schema family which needs a special mention here. It is called "Generic models of dynamic systems." This schema family contains schemas for the rate of change of the values for quantities. For example, if a quantity increases linearly, then the associated schema is called linear_transfer; if the quantity increases exponentially, such as in the problem presented in figure 2.1, its called exponential_transfer. Other schemas in this family are "Goal-seeking" "Equilibrating," "Accelerating," and "Epidemic."

All the problems used in the study reported here are solved using just 3 schemas from the "generic models" family: linear transfer, exponential transfer and accelerating transfer. Thus, a model of a particular student consists of just 3 numbers, each between 0 and 1, representing the probability of master of each schema. It may seem simple to keep these three numbers up-to-date as the student works, but, as will be shown below, it is anything but simple.

It is common for Dragoon problems to require multiple schemas for their solution. When two schemas are applied to the same problem, they must share at least one node. When a node belongs to two schemas, it is not clear how to interpret the steps the student makes when constructing that node. This makes student modeling quite difficult, as will be seen in a moment. Figure 2.4, shows node *bus fleet* which is part of more than one schema that is linear growth and exponential decay.

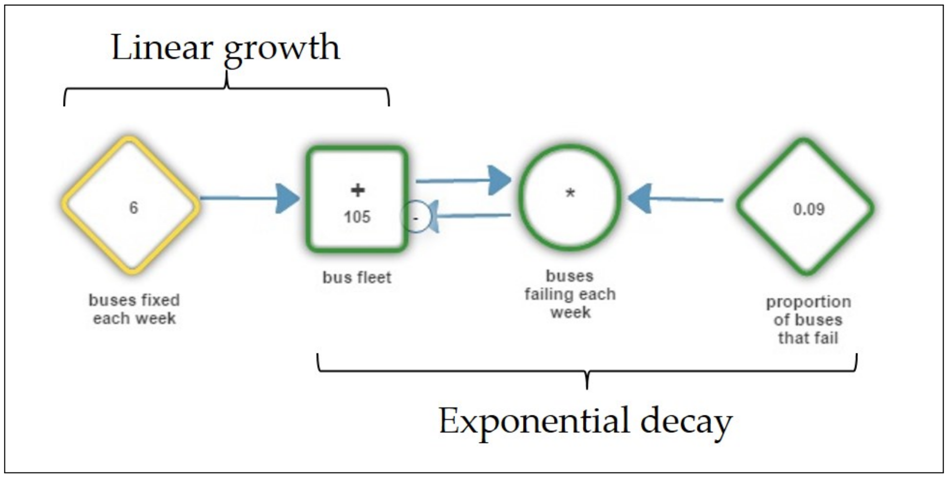


Figure 2.4: Schema Overlap in Dragoon in Node *Bus Fleet*

Another problem is that schemas are a crude representation of the difficulties that students face when reading and understanding the description of the system. For instance, there may be “clue words” in the system description that strongly suggest which schema needs to be applied. Here’s a second issue: although noun phrases are used for most quantities in a model, sometimes quantities are only implied by the system description and not mentioned explicitly. Thus, a problem may have difficulty or easiness that is not reflected in the set of schemas needed for solving it. This is another challenge for student modeling in the Dragoon context. These difficulty parameters are defined by author while constructing a problem and defining schemas applied in it. Figure 2.5, shows the interface to define a schema and the checkboxes represent whether the schema application has those difficulty parameters applicable or not.

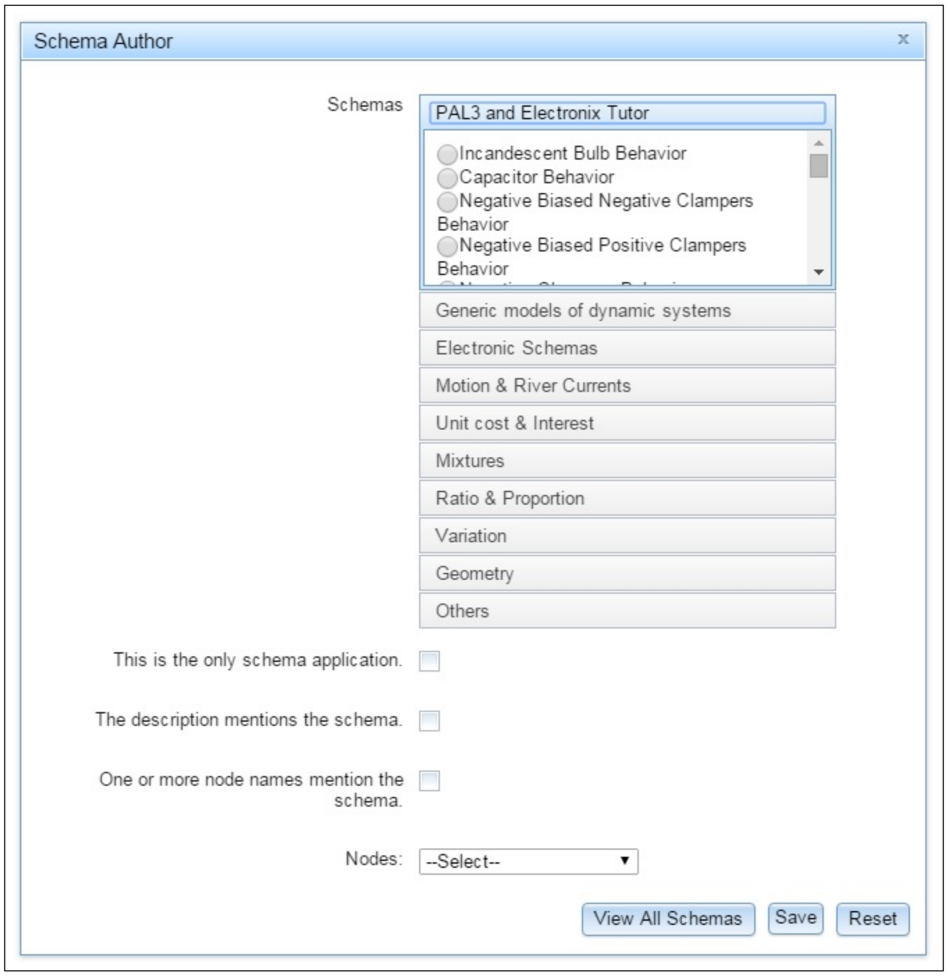

Schema Author
Schemas
PAL3 and Electronix Tutor
Incandescent Bulb Behavior
Capacitor Behavior
Negative Biased Negative Clampers Behavior
Negative Biased Positive Clampers Behavior
Generic models of dynamic systems
Electronic Schemas
Motion & River Currents
Unit cost & Interest
Mixtures
Ratio & Proportion
Variation
Geometry
Others
This is the only schema application.
The description mentions the schema.
One or more node names mention the schema.
Nodes:
--Select--
View All Schemas
Save
Reset

Figure 2.5: Interface for Author to Define Schemas with Difficulty Parameters

Thus, given our description of Dragoon and the associated schema, the research problem can be written as: For each schema in Dragoon, maintain a number which defines the probability of mastery by the student for that schema, based on the right or wrong entries given by the student as they construct nodes.

Chapter 3

GRAPHICAL MODELS AND STUDENT ASSESSMENT

This chapter examines the problem of creating a student assessment. It first reviews one of the techniques which is used for assessment and then elaborates on it's application for in Dragoon. It assumes that the reader is somewhat familiar with Bayesian Networks and probability.

3.1 Dynamic Bayesian Networks (DBN)

DBN (Russell and Norvig, 1995) are Bayesian networks that represent the changes in a system over time by duplicating the network once for each time slice. The notation for representing such random variables is to write X_t which denotes the variable's value at time slice t

In the type of DBN used in student modeling, there are two types of variables (see figure 3.1):

1. State variables (X_t): These define the state of the domain at time t, which is assumed to be the current time.

2. Evidence variables (E_t): These are the observations which are conditionally based on the current state variable X_t.

These DBNs are based on the Markov assumption, which assumes that the whole history of the process up to this point is summarized in a state variable. Thus, the evidence at time t can be adequately predicted from X_t and knowing the values of the earlier state variables provides no added information.

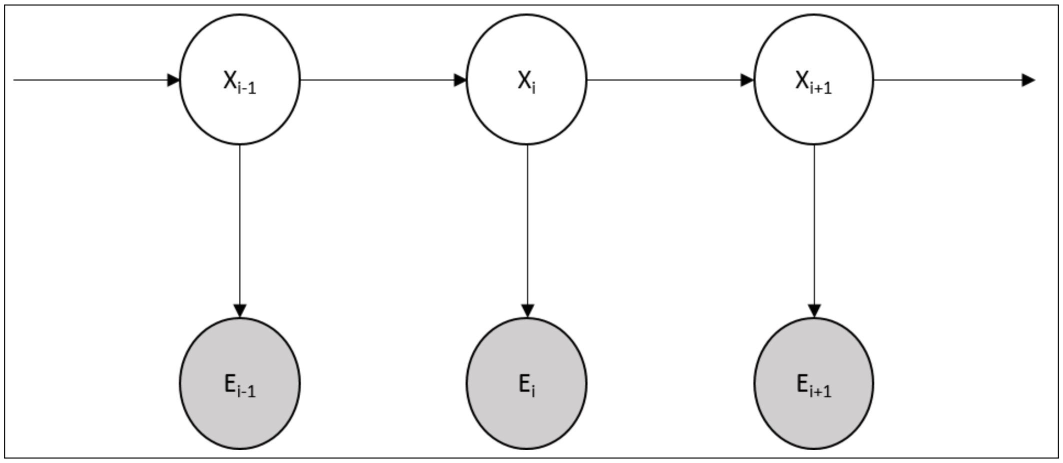


Figure 3.1: Dynamic Bayes Network with X Hidden Variables and E Evidence Variables

In student modeling, we define state variables $k_{q,t}$ where q is the skill and t is the current time. Every such variable has two values, "mastered" or "not-mastered." From here on, "skill" will be used for a piece of knowledge that can be learned and applied independently of others. In the Dragoon context, a skill is a schema. The evidence variables y_t, represent whether a student gets a step correct or incorrect.

Plate notation is a handy way of portraying a DBN. It shows the parts of the network that are duplicated at every time slice inside a rectangular plate, whereas parts of the network that are not duplicated are shown outside the plate. The DBN of Figure 3.1 is shown in Figure 3.2 using the plate notation. The state variables X_t have been replaced by $k_{q,t}$ and emission variables have been replaced with y_t to show the DBN used for student modeling.

As usual in BN notation, each node represents not only a random variable but also a conditional probability table, where the conditioning variables are indicated by the incoming links. Thus, the $k_{q,t}$ nodes have a conditional probability table in them

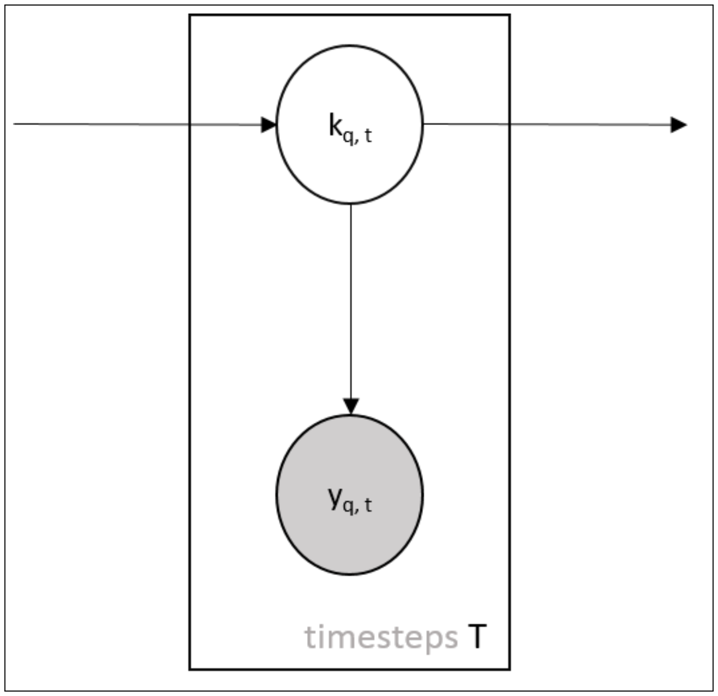


Figure 3.2: A DBN for a Skill Used in Student Modeling Using Plate Notation

called the transition probabilities, and the evidence nodes have a table in them called the emission probabilities.

3.2 Inference in Graphical Models

There are several common types of probabilistic inference used with DBNs, and each has algorithms for carrying out the computations. Some of these inference problems in DBN are used in student modeling:

1. Filtering: the process of finding the posterior distribution over the most recent state given the complete emission sequence, that is calculate $P(X_t|e_{1:t})$. That is, given the students' history of correct and incorrect steps up to time t, calculate the probability of mastery of each still at time t.

2. Prediction: the process of predicting the distribution after k epochs from the current time given the emission sequence till now, that is calculate $P(X_{t+k}|e_{1:t})$. Not currently used in student modeling.

3. Smoothing: Given the emission sequence until time t, calculate $P(X_{t-k}|e_{1:t})$ for all k between 1 and t. This is used during the EM procedure (see below) for calibrating a DBN given data from students.

4. Most likely explanation: the process of calculating the most likely sequence of the state variable values given the emission variables, $argmax_{X_{1:t}}P(X_{1:t}|e_{1:t})$. Not currently used in student modeling.

3.3 Calibrating DBNs

Calibrating a DBN means figuring out the conditional probabilities (i.e., parameters) that define the network. The basic technique to find the model parameters θ is to calculate

$$argmax_{\theta} \prod_{D} P(D|\theta) \tag{3.1}$$

where, D is the data produced by the network in the form of all the observable values and θ are defined as the conditional probability values for the network. The product of probabilities is called the *likelihood* of the data given the network parameter values. Since it is a product of probabilities (which are bound to be less than 1 and their product is even smaller), we usually use the log values of the probabilities as a maximizing function. It is called the log likelihood of the data. It is given by equation 3.2

$$L = \sum_{D} log(P(D|\theta)) \tag{3.2}$$

Because DBNs have latent (hidden) variables whose values cannot be observed directly, so calibration is usually done using a greedy search called Expectation Maximization (EM). It starts by assign random values to the conditional probabilities, θ. Then their values are improved by repeating the following two steps until improvement is negligible:

1. Expectation step: Smoothing. That is, we calculate the probabilities of mastery of every skill at every time given the data, which consist of the correctness of the students' steps at every time. This uses θ_i, the current values for the parameters.

2. Maximization step: Calculating θ_{i+1}, by probability-weighted counting. That is, we now have lots of slices, each with a probability of mastery of the skills and an observation of the student's step. We can update the emission probabilities by seeing how frequently each observation occurs with each knowledge state, weighed by the probability that the state occurs. Similarly, we can estimate the transition probabilities by counting how often a subsequent knowledge state occurs with a given prior state, weighted by both their probabilities.

3.4 Bayesian Knowledge Tracing

Bayesian Knowledge Tracing (Corbett and Anderson, 1994) is a simple DBN that has been widely used for student modeling. It assumes, just as we did above, that every skill has just two states: mastered and not-mastered, and the job of student modeling is to infer the student's probability of mastery for each skill given the student's behavior so far. It also assumes that every step is either correct or incorrect. Most importantly, it assumes that every step involves just one skill, and that every skill q has just five parameters that define its transition and emission probabilities:

1. Initial knowledge: $P(k_{q,0})$, knowledge of a student at time $t = 0$.

2. Learn: $P(k_{q,t} = mastered|k_{q,t-1} = notmastered)$, what are the chances that student has mastered the skill q at time t given that the skill was not mastered at time .

3. Forget: The converse of learning is forgetting, which is given by $P(k_{q,t} = notmastered|k_{q,t-1} = mastered)$. Many models assume this value to be 0.

4. Guess: $P(y_{q,t} = correct|k_{q,t} = notmastered)$, chances that student answers correctly even if the he has not mastered the skill.

5. Slip: $P(y_{q,t} = incorrect|k_{q,t} = mastered)$, chances that student answers incorrectly even if he has mastered the skill.

Here learn, forget are the transition probabilities and guess, slip are emission probabilities and initial knowledge is the initial state of the student.

3.5 Feature Aware Student Knowledge Tracing (FAST)

FAST is a recent extension of Bayesian Knowledge Tracing (González-Brenes et al., 2014). It replaces all five parameters with logistic regressions, based on the features available for them. That is, in addition to being able to observe whether a step is incorrect, it is assumed that we can observe the values of a set of features, such as the difficulty of the step, whether the student paused a long time before responding, and so on.

For example, suppose that we only need two features to adequately model a certain skill, call them $f_1(t)$ and $f_2(t)$, where t denotes a step, indexed by its sequential time of occurrence. These functions are binary, in that they return 1 if the feature is present at step t and 0 if it is absent. Thus, instead of five constant values for Initial knowledge, Learn, Forget, Guess and Slips, we have:

- $Init(t) = logistic(I(t))$ where $I(t) = b_{i,0} + b_{i,1} * f_1(t) + b_{i,2} * f_2(t)$
- $Slip(t) = logistic(S(t))$ where $S(t) = b_{s,0} + b_{s,1} * f_1(t) + b_{s,2} * f_2(t)$
- $Guess(t) = logistic(G(t))$ where $G(t) = b_{g,0} + b_{g,1} * f_1(t) + b_{g,2} * f_2(t)$
- $Learn(t) = logistic(L(t))$ where $L(t) = b_{l,0} + b_{l,1} * f_1(t) + b_{l,2} * f_2(t)$
- $Forget(t) = 0$ (inherent assumption of FAST)
- $Logistics(x) = 1/(1 + e^{-x})$

Note that FAST still retains the assumption of Bayesian Knowledge Tracing (BKT) that each step involves applying one and only one skill. It also retains the assumption that parameter values are the same for all students over all steps, but different skills can have different values for their parameters. With regards to allowing the transmission and emission probabilities of BKT to vary across steps, different authors have made different assumptions. Most BKT applications assume Learn and Forget have the same value for all steps, but slip and guess can have different values for different steps. FAST assumes that the regression parameters are the same for all steps, and represents the variability due to steps by varying values output by the features.

In order to find values for the parameters, both BKT and FAST typically split data by student. For instance, given data from 100 students, they might calibrate (i.e., find parameter values that best fit the data) against 90 students then test the resulting model's accuracy by predicting the behavior of the remaining 10 students. This way of splitting the data into training and test sets relies on the assumption that parameter values are the same for all students, and it reflects the way the models are used in practice, which is to calibrate on an initial sample of students then use those parameter values with all subsequent students.

In the example above, we have provided more parameters to the model, namely the 15 coefficients in the five regression equations. They replace the 5 parameters of BKT. This means that we can expect FAST models to fit the data better, but there is a risk of overfitting unless we have a huge amount of training data for calibration. Fortunately, the EM algorithms can be modified to find values for regression coefficients. The basic idea is the replace the M step with a standard gradient descent method used for calibrating linear regression equations (Khajah et al., 2014).

3.6 BNT-SM

BNT-SM (Xu and Mostow, 2010) is similar to FAST in that it replaces constant parameter values with logistic regressions. However, unlike FAST, it does not use regularization to prevent over-fitting the data for logistic regression. Regularization is the process of adding a penalty to variable values so that they do not overfit the data. For example, if w are the weights that are being minimized, then the minimization function is updated to add a penalty term $\alpha * ||w||$ where α is the regularization parameter. This keeps a check on the value of w.

Like FAST, BNT-SM also uses EM for calibration.

Chapter 4

STUDENT MODELING METHODS FOR DRAGOON

This section describes the several student modeling methods that were developed for Dragoon. Their evaluations are described in subsequent sections.

As mentioned earlier, the skills in Dragoon correspond to schemas, and there are only 3 schemas used in the studies described below. The schemas have between 2 and 4 nodes. Moreover, a step corresponds to entering a property of a node.

The technical challenge arises in meeting the assumption of BKT (and FAST and BNT-SM) that every step addresses just one skill. When a student is solving a simple problem that just requires applying one schema, then this assumption is met trivially. The challenge occurs when two or more schemas are required, and the student is entering a node that belongs to more than one schema.

To meet this challenge, we defined skills corresponding to schema combination. Whenever it was possible for a node to belong to two schemas, S1 and S2, then we defined a new skill S1_S2 that represents the competence of knowing how to construct a node that is shared between the two schemas. Now sometimes two schemas can share different kinds of nodes, so we actually defined two different extensions:

- Schema-type: A skill is defined to represent the combination of schema together with the type of node. For example, if an accumulator node is shared between the linear and exponential schemas, then it is covered by the skill exponential_linear_accumulator, whereas exponential_linear_parameter covers the case of a parameter node being shared between the two schemas.

- Schema-only: S1_S2 represents skill at constructing any node shared between S1 and S2. For example, if an accumulator node is part of both a linear and an exponential schema, then the skill will be exponential_linear. If a parameter node is part of both a linear and an exponential schema, then it too would be covered by exponential_linear.

The main difference between the two is how students understand model construction while interacting with Dragoon. Do students think in terms of schemas or in terms the type of node in those schemas? In other words, is each node type part of different skill or not?

The experiments below compare these two different ways of extending the basic set of 3 skills.

4.1 Subskills

When editing a node, the student must enter its description, type, initial value, units and expression. Each type of entry is a step, but they are vastly different in terms of their difficulty, the chance of guessing correctly, and the chance of slipping. Thus, it makes sense to use the features of FAST and BNT-SM to differentiate them. This would essentially allow the different types of step to have different values to represent slips, guesses, learning and forgetting. Similarly, accumulator nodes are more complicated to define than function nodes, which in turn are more complicated than parameter nodes, so perhaps we should have features distinguishing them as well. In the literature, this use of features is often referred to as defining "subskills" of the skill.

Recall that we have two extensions (schema-only, schema-type) to the original set of 3 skills (linear, exponential and acceleration). For every extension, subskills can be defined in three ways:

- While constructing a node, the student must input five properties (Section 2.1); each property becomes a sub-skill for the skill. For example, linear_units is the subskill for a student choosing the units of a node that is part of the linear schema.

- The type of a node (function, accumulator, or parameter) is the sub-skill for that schema. For example, linear_accumulator is the subskill for an accumulator node which is part of the linear schema. In this case, a node belongs to more than one schema and constructing the node requires multiple sub-skills.

- The type of node together with one of the five properties (Section 2.1) is a subskill for that schema. For example, linear_accumulator_type is the sub-skill for the student choosing the type of a node that is an accumulator and is part of the linear schema. This is the most specific subskill model.

The final model that we define is the ”Control” model; this model has no knowledge of the schemas. Instead, the skill is just to create a node of type accumulator, function or parameter and the associated five sub-skills are defined to be the five properties (description, type, initial value, units, and equation) of those nodes.This model corresponds to the case where students are not thinking in terms of schemas and “mechanically” completed the nodes. We conjecture that this corresponds to a shallow way of learning to model and when students eventually start to author models of dynamic systems, they would encounter difficulties to create the basic structure of the models.

Table 4.1 shows the models possible with the combination of skills and sub-skills. It also gives them a name which will be used to present the results in the next chapter.

Skills \ Subskills	**Schema_nodeType_property (31)**	**Schema_nodeType (8)**	**Schema_property (14)**
Schema and node type (15)	Group1	Group2	Group3
Schema (9)	Group4	Group5	Group6

Table 4.1: Skill and Subskill Combination Tried for Model Fitting in Dragoon. The Numbers in the Bracket Show the Count of Skills and Subskills in the Data. "Control" Model Was Also Part of the Results Group.

4.2 Difficulty

As mentioned earlier, the way the text describes a system can make it harder or easier to construct a model of the system. An even more accurate model of student's performance could perhaps be had if we associate a difficulty feature with each problem. To test this hypothesis, 2 different models were created with Difficulty parameters.

Originally in Dragoon, we define difficulty parameters for each schema and not for the problem. Figure 4.1 shows the DBN which is based on this model. This was the first model created to define difficulty parameters. Since each schema had its own difficulty parameter, the steps where there were multiple schemas involved was assumed to be separate steps with the similar student response. First model was created by adding this separated model of difficulty to transition features model of sub-skills.

Since, IRT model defines difficulty to effect the student emission probability, so the difficulty parameter definition was updated to make it same parameter for the complete problem. If any of the schema had the difficulty parameter value set to 1, then it was assumed to be 1 for the complete problem. This new model, would have ensured that we do not need to separate the steps to different steps when multiple schemas were active while constructing a node, as was the case in the first model. Second model, was actually based on the results from previous FAST model fitting parameters. To check if difficulty parameters had an effect on the parameters already calculated, so they were added to the model with the better AUC (Area under the curve) value.

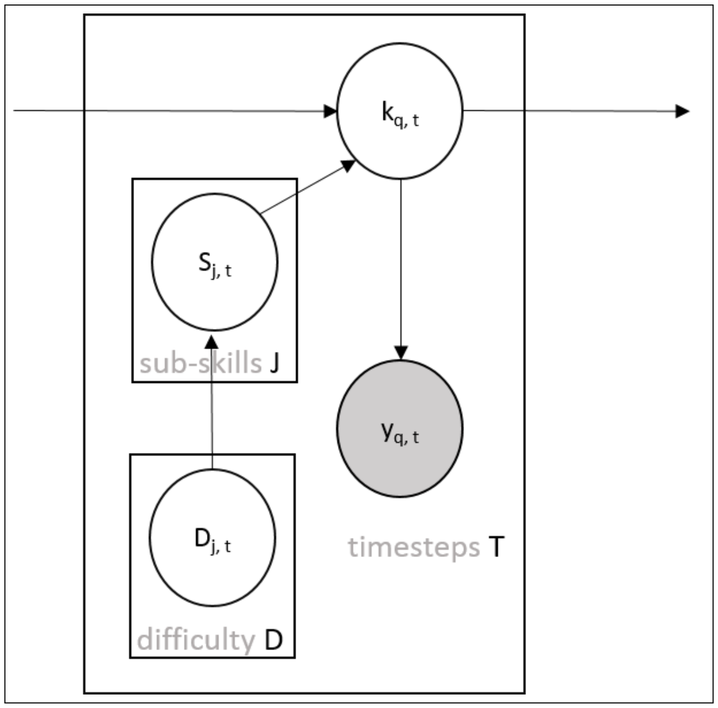


Figure 4.1: DBN with Difficulty Parameters for Sub-Skills

4.3 Adding Features to Emission vs. Transition Probabilities

FAST allows one to add features to any of the five BKT probabilities. However, González-Brenes et al. (2014) remark that it suffices to add features to either the emission probabilities (slip and guess) or the transition probabilities (learn and forget), and that adding features to both types does not usually improve model fit. Because González-Brenes et al. (2014), found that adding features to the emission probabilities gave somewhat higher accuracy than adding them to the transition probabilities, we decided to add sub-skills to emission features as well. The original model of sub-skills as transition features was proposed by Xu and Mostow (2010), where they had added subskills as transition features only.

4.4 Implementation Detail

Rather than use FAST for all our experiments, we decided to use BNT-SM and FAST to fit the models. The most important difference between the two is that FAST uses regularization to prevent overfitting of the data.

Implementation of BNT-SM is largely dependent on Bayes Nets toolbox (Chang et al., 2006). It needs three files:

1. Training data
2. Testing data
3. Property XML

Training and testing data are provided in tabular files which include skills, subskills, user, and the observed student response. These are used to find and test the parameters of the LR-DBN model. The difference between the two is that BNT-SM calculates the chance that the student's response would be correct for the training data. These values will be used to calculate the ROC curve.

Property XML defines the DBN created and which column corresponds to the node defined in the data files. It has four important parts. They are:

1. Input: This structure has the names of the training and testing file.
2. Output: This structure has the out files for analysis, such as the log file, parameter file, and inference results.

3. Nodes: These have all the nodes that are present in the LR-DBN for the first time slice only. LR-DBN refers to the implementation of Dynamic Bayesian Networks using logistic regression. This is the implementation that BNT-SM toolkit uses. The properties needed to define the node completely are: the column which represents the value in data files, whether the node is observable or latent, the connection to different nodes using “within” and “transition” tags. For example, a knowledge node is latent so its value in the data column was set to “NULL;” this column name is added to the node structure including a connection to subskills and student outcome using the “within” tag and to knowledge using the “transition” tag.

4. Eclasses: They have all the probability values for each connection. To define this completely, exhaustive conditional probability variables are needed for every node and parent node pair. The values are defined to be random based on the starting value of 1 - probability value already defined.

The Inference result file was used to calculate the ROC curves and the AUC values.

FAST must be supplied with three files:

1. Training data

2. Testing data

3. Configuration

First two files are same but the column names are rigid in this case as there is no XML file to describe these details. The structure of the nodes are fixed as well. Two types of features are possible to define using configuration file, the transition and emission features (Huang, Y., et.al, 2014). If a node is connected to knowledge, then it is part of the transition features; otherwise, if a node is connected to a student response node, then it is part of the emission features. Figure 4.1 shows that subskills are part of the transition features. This was used to implement two different types of DBN:

- Subskills: It was similar to the DBN created for BKT. Knowledge/skill node was made up of multiple sub-skills. Figure 4.1 shows the DBN structure where only subskills were added to the model. The multiple subskills were added as transition features (called emission constant models) and emission features (called transition constant models) using the configuration files.
- Difficulty parameters: as explained in section 4.2, different models were created where they were added as emission features to the already created subskills models for FAST.

4.5 A BKT Model with Subjective Probabilities

Dragoon has 10 schema families and 108 schemas in them. Currently, 40% of these schemas do not have application problems where students can learn to apply them for modeling. Moreover, the data that we collected was for 3 schemas which fell under 1 schema family. So to create an embedded assessment using Bayesian knowledge tracing for schemas which are not exactly part of the data, one needs to define the parameters using a different process.

In order to apply BKT when calibration data is not available, it is customary to estimate the emission and transition probabilities. For example, one might define Guess to be 1/N for multiple choice solution with N answers. When a human guesses a parameter value that is a probability, it is customary to call the value a "subjective probability." So this section describes a BKT model with subjective probabilities instead of probabilities developed via calibration. This subjective probability model was used in only one of the two evaluations described later, namely the evaluation with simulated students. This section describes how the values for guess, slip, learn, forget and initial probability of mastery were chosen.

The guess parameter is the probability that the student answers a step correctly given that they have not mastered the required knowledge. In Dragoon, this value can be calculate for every property of a node. The values are:

1. Description - If there are total n nodes in the problem out of which, there are m nodes which are not part of the solution model, then guess value for this is $(n-m)/n$.

2. Type - its value is always 1/3 as we have three options in type dropdown out of which one of them is correct.

3. Initial Value - if there are n parameters and m accumulators in a problem then the guess parameter value for this is $1/(m+n)$.

4. Units - if there are n different units for the node then the guess value will be $1/n$.

5. Equation - this is based on the number of nodes in the expression and the operator used between them. If there are 2 nodes in the equation and 1 operator of "+" in between them and there are n nodes in a problem. Then the guess parameter is given by $(2/n) * (2/n) * (1/4)$. If it is using function other than the basic arithmetic operators like "max" then Dragoon has 29 such different functions and there are m different nodes in the operator thus the chance guessing the answer will be given by $(1/29) * (m/n)$. The actual value here was $(1/29) * ((m/n)^m)$ but this value was going very small and was causing a

very high increase in the student learning. To make the transition smoother the exponent was removed.

The slip parameter is the probability that a student who has mastered the requisite knowledge for a step nonetheless answers that step incorrectly. The slip parameter essentially defines what mastery means - a low value for slip means that "masters" are highly accurate and rarely make mistakes, whereas a high value means that "masters" often makes mistakes and yet their competence (or lack thereof) is still acceptable for "mastery."

Even though the slip parameter's value is partially a pedagogical decision (i.e., what probability of error are we willing to tolerate from our "masters?"), it is also affected by the complexity of the step. For instance, mistakes when entering an equation are more likely than mistakes when selecting a type from a three-item menu.

Our idea for calculating slip values was taken from d Baker et al. (2008), where they use the number of correct and incorrect solution history of a student to calculate values for the slip parameter. Our method starts every schema with a slip value of 0.05. An increment of 0.002 was added to the slip parameter for the next time slice, if the student's current action was correct. There was an upper bound of 0.1 set to this value. If the student saw the hint for a property then the slip parameter was increased by a factor of 0.01. All the calculations were done only for the nodes in the schema and not for the complete problem.

The initial probability of mastery was set to 0.1, the learn parameter was set to 0.05, and the forget parameters was set to 0. These probabilities are something that would be added to schema definition as they depend on the author of the problem, how much initial knowledge was provided. For now, for the testing purposes these values have been used.

This student model assumes that learn and forget have the same value for all students, all problems and all times. The guess parameter has different values on different problems, but has the same value for all students at all times. However, the value for the slip parameter is allowed to vary across students, problems and times. If the student is learning, and in the process he is entering correct responses, so to give him credibility we increased the slip probability to recognize that his mistake could have been an error.

Probability of mastery was defined at the level of schemas, where a schema is defined as a group of nodes. On the other hand, the incorrect/correct data comes from steps (student actions) which occur at level of node properties. So the implementation had to take care of transferring the values from node properties to schemas.

The method used for this was to do a weighted sum over the change of knowledge value for node properties and then do a weighted sum for each node to see the change in the schema probability value. The weights were defined in the order equations $>$ type $>$ units $>$ initial value $>$ description. For combining nodes the order of weights was defined as accumulator $>$ function $>$ parameter. Finally the probability value was added from node properties to single number representing the mastery of schema using these weights. This granularity would be fit for suggesting problems to students.

Chapter 5

RESULTS AND DISCUSSION

Previous chapter presented, some basic topics and implementation using FAST and BNT-SM. This chapter presents the data collection and the results on the data. The last section discusses the results and compares them to bring out various observations from them.

5.1 Data collection

One way to evaluate a student model is to measure it's fit to a collection of student data. That is, part of the data are used to calibrate the model, then the accuracy of its prediction of the remaining data is measured.

For this evaluation, data were collected during a lab study in Summer 2015. Participants were a mix of undergraduate and graduate students at Arizona State University with a good background in Mathematics (at least high school algebra). They were not asked for a background in modeling dynamic system for the study and only their mathematical background was inquired. Students had to come for one session where they had to finish fourteen Dragoon problems. There was no upper limit to the time it would take. But on an average students were able to finish the study in 2 hours 11 minutes. Students were compensated with money for their work.

Two workbooks were created to teach students dynamic modeling. The first workbook was for introducing Dragoon's interface to students with its basics, like how values of quantities are changing. There were three Dragoon problems in this workbook, with some multiple choice questions as well. The answers to multiple choice questions were not logged. The second workbook introduced students to the concept of schemas and how the values in dynamic models changed. The second workbook then gave students ten Dragoon problems to solve. The order of first seven problems were chosen randomly and last three were fixed. Although the problems could be arranged by the level of difficulty, the problem order was randomly chosen to understand, the role schema difficulty parameters played while students solved the problems. Next three problems were always given in a fixed order. In table 5.3 you can see that they were huge problems and with many schema applications, thus making them very difficult and were not part of the randomly chosen problems. After students finished the 10 workbook problems, they were assigned an eleventh problem which was done in delayed-feedback mode. This was like a post-test problem for Dragoon, to understand if the students understood modeling. This was followed by a questionnaire to ask whether they think they understood modeling any better and if the study was useful in helping them.

We could not use the post-test problem in analysis as 47 out of 52 students did the problem correctly. This ceiling effect led us to concentrate on other empirical testing of using model fit to define its accuracy on test data.

52 students took part in the study; the log data for the student work was collected. It had a total of 9336 steps, where each step involved making an entry in the node editor. There were two primary schemas taught by the second workbook: linear transfer and exponential transfer. In addition, there was a third schema, accelerating transfer, which was not explicitly presented to the students in the workbook. Nonetheless, it was present in two out of the ten workbook problems. These two problems were given to the students towards the end of the study. Table 5.1 shows the number of solution steps where a student could apply a given schema. Students were also asked to send the screen shot of their response for questions in workbooks. These responses were not used in any way for the results.

Schemas	Application Opportunity Count (per student)
Linear transfer	86
Exponential transfer	115
Accelerating transfer	42

Table 5.1: Schema Counts per Student While Interacting with Dragoon.

Schemas	Application opportunity result count	
	Correct	Incorrect (percentage)
Linear transfer	3971	722 (15.38)
Exponential transfer	4184	848 (16.85)
Accelerating transfer	1668	497 (22.96)
Total	7798	1538 (16.5)

Table 5.2: Total Count of Schema Application Opportunity Distribution for Correct/incorrect Student Response.

Distribution of correct/incorrect responses for each schema is shown in Table 5.2. If a node belongs to more than one schema, then it was counted as a separate schema application opportunity for both of them. Thus, the total application count does not match the sum of all schema application numbers presented in the table.

Table 5.3 shows the number of nodes present in each of the ten problems. The number on the right does not show the sequence of problems. It is rather one of the sequence that a student had got. First seven problems were given randomly followed by three problems which were in a fixed order. Each problem consisted of two to three schemas with the last problem had seven instances of schemas.

Problem #	# of Nodes	# of Schemas
1	5	2
2	4	2
3	3	2
4	3	2
5	5	3
6	6	3
7	3	2
8	5	2
9	4	2
10	12	7

Table 5.3: Number of Nodes and Schemas in the Problems Used for the Study.

5.2 Results using BNT-SM

As described earlier, we defined seven different assignments of skills and subskills (see Table 4.1) and used two different modeling paradigms, a version that includes only subskills (BNT-SM & FAST) and a version that includes subskills and difficulty parameters (FAST). This section presents the results from fitting BNT-SM to the data, and the next section presents results from fitting FAST to the data.

The BNT-SM model was calibrated separately for each of the seven definitions of skills and subskills, then each of the seven calibrated models accuracy was tested separately. The data were divided into 80% for training and 20% for testing.

The models did around random chance (AUC value around 0.5). The values guess and slip parameters for the schema and skill were either 0 or 1. Table 5.4 shows the AUC value for all the skill and sub-skill model defined in the previous section.

Skill model name	**AUC**
Group1	0.510
Group2	0.503
Group3	0.504
Group4	0.491
Group5	0.504
Group6	0.516
Control	0.515

Table 5.4: AUC Results for BNT-SM.

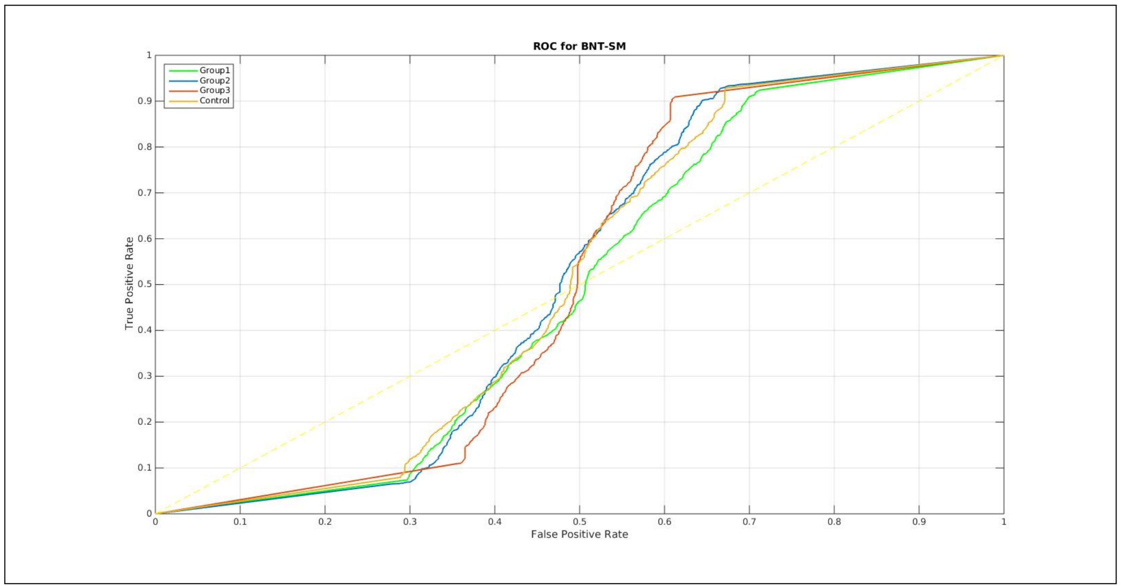


Figure 5.1: ROC Curve using BNT SM

As observed from the ROC curve we can see that the results are poor. There was a very high amount of false positives for which the probability of mastery was calculated to 1. This result showed that the BNT-SM was over-fitting the data. Thus it was not used anymore as all the variations that were tried were also over-fitting the data, due to which the values were not added to the table as well.

5.3 Results using FAST

Although this model uses subskills in logistic regressions for the learn parameters, just as BNT-SM did, subskills were also tested as logistic regressions for emission probabilities (guess and slip). Then difficulty parameters were also added to the emission model of subskill.

The data was divided into 80% (42 students) for training and 20% (10 students) for testing. Like in BNT-SM a single set of parameters for each skill were found from the complete dataset.

This algorithm showed better results than BNT-SM. Table 5.5 shows the AUC with subskills emission constant features. Table 5.6 shows the AUC results with subskills as emission features. The results are shown for all the groups presented in section 4.1 as well as the control model. The curves are presented on the test data. There is no overlap in training and testing data and the predictions were made over all student observations in the test dataset.

The parameters are initial knowledge, learn, slip and guess. To get the count of parameters in say, Group1 for transition features model over subskill we will have to use the table 4.1. It has 15 skills, so for each skill had 4 parameters (init, learn, guess and slip; forget is 0) would mean 60 parameters in total. But when we assume transition features, then 45 parameters are constant. The transition probability (learn) is defined using logistic regression over subskills. It depends on the subskills that can be active for a skill at any time, for example linear_accumulator will depend on five subskills (for each step) and thus 6 parameters will replace the transition probability.

The increase in parameters from transition features model to emission features model is due to the fact that, FAST assumes forget probability to be 0. Table 5.7 shows addition of difficulty parameters to the subskill as an emission feature, which adds six parameters (three for each, guess and slip) for every skill in the group. This increased the feature count by 90 for Group1, Group2 and Group3 skill models and by 54 for Group4, Group5, Group6 models. This has left some of the models with too many parameters where there are up to 36 emission features for one skill and thus leading to an underflow situation. In the Control model, the repetitive values of the same skill and subskill created an infeasible problem for optimization using LBFGS (due to repetitive values). Such scenarios are shown in the table with value "NaN".

Figure 5.2 and 5.3 show ROC curves for FAST for emission constant model. Figure 5.4 and 5.5 show the ROC curves for FAST for transition constant models. The curves are completely distinguishable as shown by the AUC values as well.

Skill model name	**AUC**	**# of Parameters**
Group1	0.700	116
Group2	0.699	81
Group3	0.698	132
Group4	0.702	132
Group5	0.684	132
Group6	0.686	132
Control	0.695	28

Table 5.5: AUC Results for FAST Algorithm with Subskills as Emission Constant Features

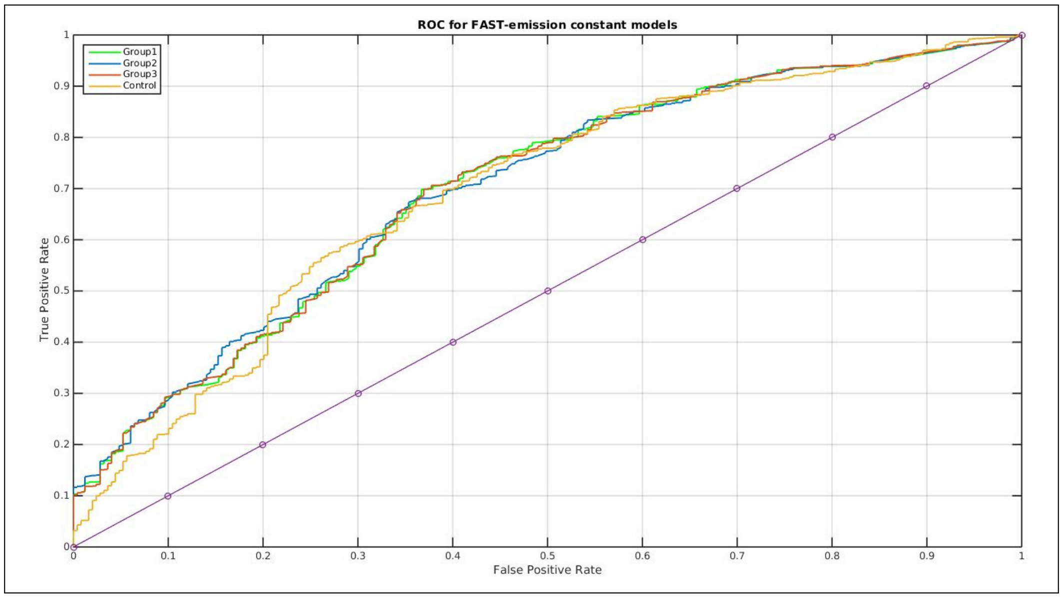


Figure 5.2: AUC Results for FAST Algorithm for Emission Constant Model for Skill Defined as Schemas_nodeType.

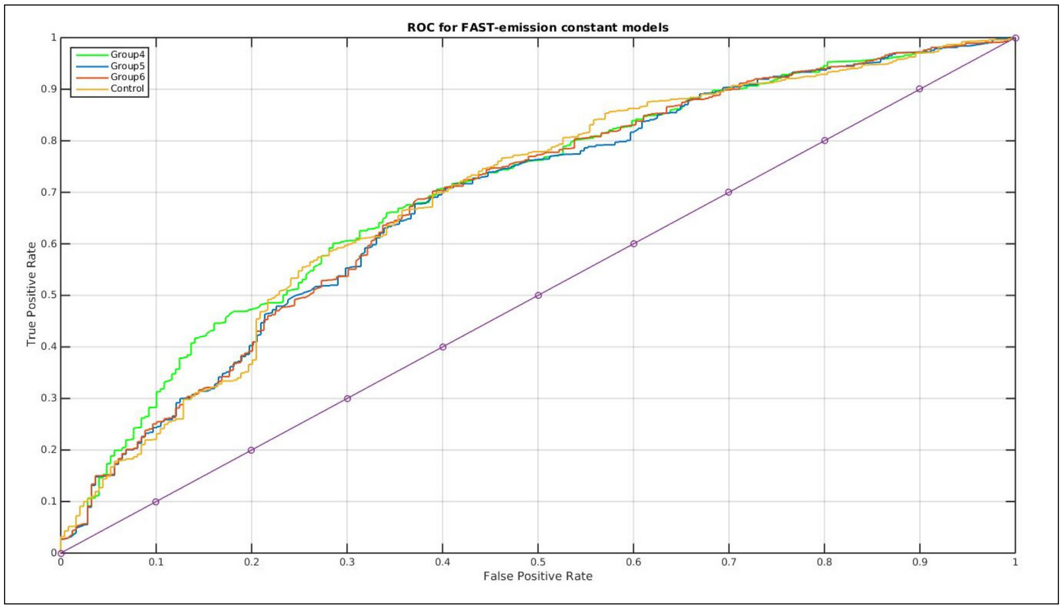


Figure 5.3: AUC Results for FAST Algorithm for Emission Constant Model for Skill Defined as Schemas.

Skill model name	AUC	# of Parameters
Group1	0.831	172
Group2	0.698	102
Group3	0.832	214
Group4	0.836	160
Group5	0.707	90
Group6	0.837	202
Control	0.813	38

Table 5.6: AUC Results for Fast Algorithm with Subskills as Emission Constant Features

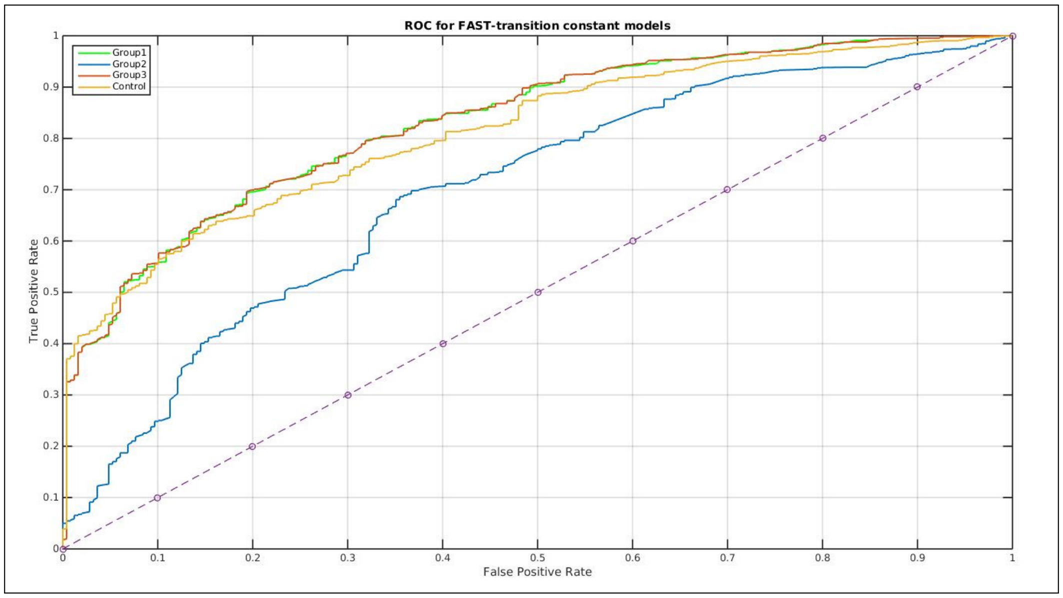


Figure 5.4: AUC Results for FAST Algorithm for Transition Constant Model for Skill Defined as Schemas_nodeType.

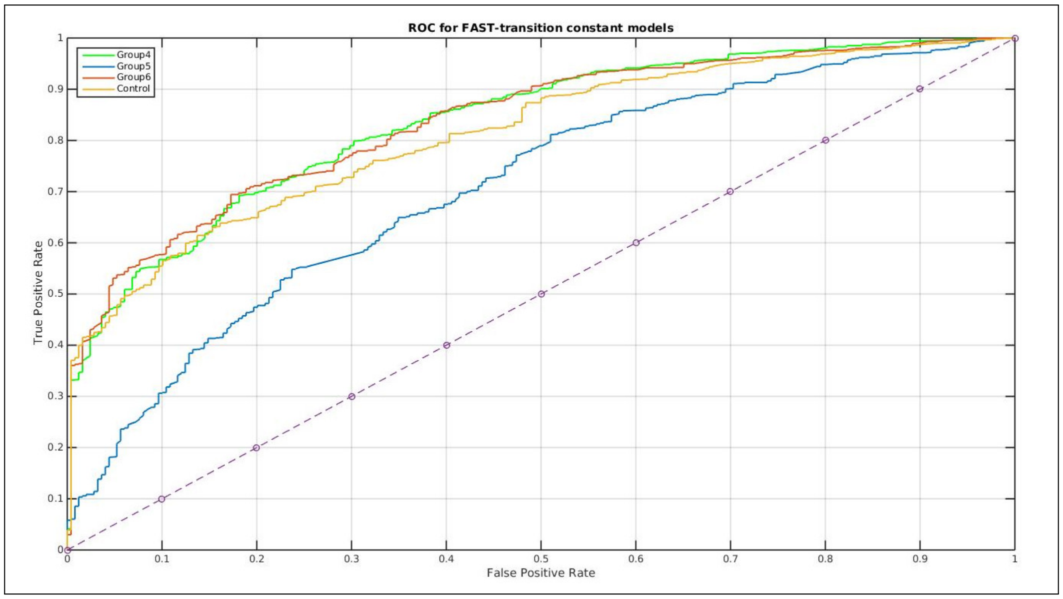


Figure 5.5: AUC Results for FAST Algorithm for Transition Constant Model for Skill Defined as Schemas.

Skill model name	**AUC**	**# of Parameters**
Group1	0.832	262
Group2	0.712	192
Group3	NaN	304
Group4	0.832	214
Group5	NaN	144
Group6	0.837	256
Control	NaN	56

Table 5.7: AUC Results for Fast Algorithm with Subskills and Difficulty Parameters as Emission Features.

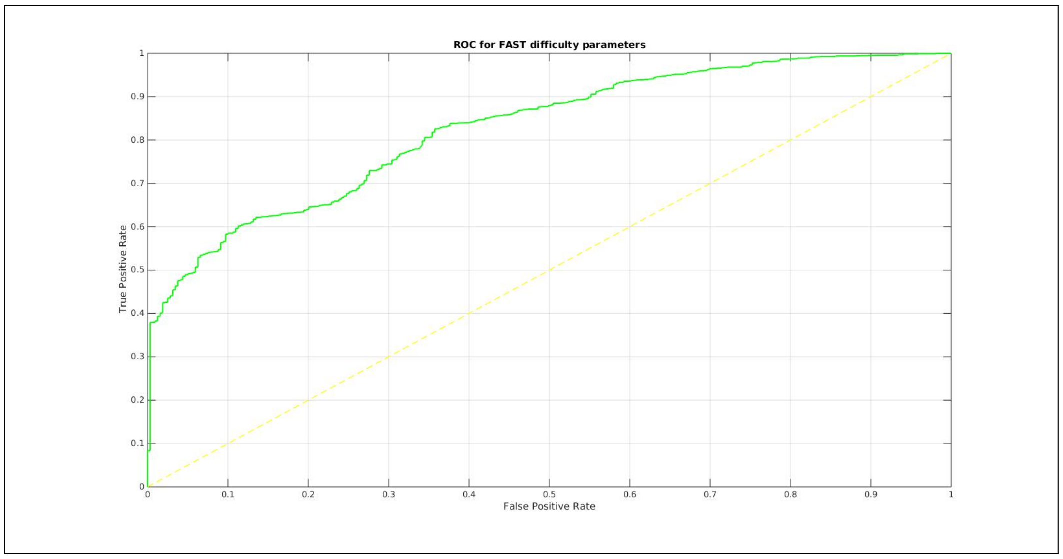


Figure 5.6: ROC Curve Using FAST for Difficulty Parameters and Subskills as Transition Features.

As explained in section 4.1, the result for the model for difficulty parameter was also calculated where each step was separated into multiple steps with the same response. This was done to use the original model (subskills as transition features) with difficulty parameters as emission features in Dragoon. It showed promising result, with AUC of 0.826. It was added to every model but it caused a very high increase in parameters and none of the models couldn't fit the data except for Group2. Figure 5.6 shows the AUC curve for Group2 model.

5.4 Comparison of Transition and Emission Subskill Models

Table 5.5 and 5.6 show the difference between the AUC values for both the models. This shows that when subskills are added as emission features they fit better to the models (González-Brenes et al., 2014). This result helps us understand that guess and slip factors for a skill are a logistic regressions over the subskills that are active for the step, and thus showing a relationship to the complexity due to node being a part of multiple schemas.

From the AUC values in transition model, one can see that all the subskill models perform at par with each other. But emission model presents a very important result which is missing in the transition features. In skill model Group2 and Group5, where subskill is defined as the complete node, AUC values turned out to be low. This gives a clear indication that we should define the subskill to be based on each step, which is to fill each property of the node.

In emission model, we can also check that the rest of the models (other than Group2 and Group5) outperform the Control model, thus showing that students did understand the schemas as the knowledge components. Thus, all our hypothesis can be completely verified by the data.

If we look deeper into the values for transition and emission features we can make a well informed guess as to why the AUC values are so different. Tables 5.8 and 5.9 provides the parameter values which were constant and not based on the logistic regression.

As we see that in Table 5.8 that very high value of guess and slip effect the model negatively, and we can hopefully infer that students were actually thinking on the modeling problems. All the models show a very similar initial knowledge for the exponential schema. This is comforting as the workbooks did have an explanation of the schema and students were taught to create an exponential models while teaching them Dragoon notation. There are other skills as well that have initial knowledge parameter of 0.62 like accelerating, which cannot be explained by the above hypothesis, as the students were given no information about this schema. But, this schema was part of the problems which were done towards the end and thus we can assume that student had learned modelling there by having a higher chance of correct results.

Skill Model	Skill	Intial Knowledge	Guess	Slip
Group1	exponential_function	0.552	0.556	0.099
Group2	exponential_function	0.543	0.556	0.099
Group3	exponential_function	0.553	0.556	0.099
Group4	exponential	0.218	0.918	0.366
Group5	exponential	0.526	0.627	0.075
Group6	exponential	0.526	0.626	0.075
Control	accumulator	0.209	0.863	0.483

Table 5.8: Parameter Values from Fast for a Skill in Transition Features Subskill Model.

Skill Model	Skill	Intial Knowledge	Learn
Group1	exponential_function	0.571	0.026
Group2	exponential_function	0.622	0.016
Group3	exponential_function	0.570	0.026
Group4	exponential	0.606	0.015
Group5	exponential	0.593	0.011
Group6	exponential	0.304	0.015
Control	accumulator	0.620	0.017

Table 5.9: Parameter Values from Fast for a Skill in Emission Features Subskill Model.

5.5 Comparing Schemas

The parameter values of guess is really high for transition subskill features model. So, it would be futile to compare parameter values for the schemas using that model. But, the emission subskill model presents an interesting result. The schemas which are made as a combination of basic schemas have a high learn rate, and the basic schemas have a high initial knowledge value. This would show that when the students started they had a good understanding of the schemas, and when complex schemas were introduced in the problems they were fluent with modeling. Table 5.10 shows some values in the Group6 skill model with emission subskills.

Skill	**Initial knowledge**	**Learn**
Linear	0.260	0.011
Exponential	0.604	0.015
Accelerating	0.617	0.016
Exponential_linear	0.127	0.020
Linear_linear	0.014	0.0058

Table 5.10: Parameter Values for Skills in Group6.

The parameter values are very similar in Group1, Group3, Group4 and Group6, showing that these models were able to fit the data in a similar pattern and thus there AUC value is also very close to each other. Other thing that can be concluded from this is that "Accelerating" was a relatively easier schema which student either got a hang of or were able to understand the problem presentation well enough to construct the model correctly.

The regression factors for emission probabilities for each subskill vary a lot throughout the result file. Thus, it is difficult to conclude anything from them.

5.6 Evaluation of the Addition of Difficulty Parameters

There were a lot of models tested to check the effect of difficulty parameters. The implementation details in section 4.2 explain the engineering required to make it work. There were few more changes needed while implementing it which were not structural but more so due to the runtime errors faced. The issues were primarily related to collinearity of the data. If we just used the difficulty parameters, then fast algorithm would end up with search step undersize error mainly due to repeated data rows which made the matrix (created for logistic regression) non-invertible. That is the user working on the same node entered the correct answer for all the properties and thus the rows were similar to the previous one. This led to adding difficulty parameters to Group2 model (subskill as transition features), which improved the AUC from 0.699 to 0.826 for the transition subskill features. A very significant change thus showing promising results.

As González-Brenes et al. (2014) point out, most studies use a fixed or nearly fixed order of problems because most instruction gradually increases the complexity of problems as the students learn more. Thus, when a model that has one difficulty parameter per problem is fit to such data, it cannot "see" the increase in difficulty because the more difficult problems are only being done by more competent students. However, our training problems were presented in random order. Thus, this is perhaps the first study to de-confound learning and difficulty, and it appears to show that adding difficulty parameters does indeed improve model fit.

Their effect was not very pronounced on the emission features model as can be seen from the table 5.7. Although during the study there were a few students which had got all the difficult problems initially and they faced a lot of trouble to do the problems, but the data is not significant enough to show that difference. This can be seen when the difficulty parameters were added as another emission feature to the emission feature subskill model in Table 5.7. It shows that difficulty parameters did not have the desired effect on the student problem solving.

As explained in section 4.2, the difficulty parameters in Dragoon are defined for each schema in the problem. When a node is part of more than one schema which causes the difficulty to be defined by more than one value. To engineer the model we needed, to break it into different steps, which is a boon as well as bane. The good thing was the comparison could be done for each schema separately, but multiple steps had led to collinearity issue as well. Thus it would not be unfair to conclude, the results improvement is promising, but it needs a change of how difficulty parameters are currently used in Dragoon. Mayer (1981) presented the idea of difficulty parameters to define the difficulty of the item based on how the problem was presented. In Dragoon, this is being used to define the difficulty for a schema for the problem. Rather it should be a part of the problem itself. This change would also be in accordance with IRT model definition of item difficulty. New proposed definitions could be as follows:

1. Schema isolation - Problem has one (easy) or more than one schema (difficult) application involved in the problem.

2. Presentation cues - If presentation explains all the schemas present in the problem (easy) or it does not (difficult).

3. Presentation noun phrases - If the presentation explicitly mentions or explains the nodes required for making the model (easy) or it does not (difficult).

There can be few more difficulty parameters, like how big is a model. If the model is bigger than say 5 nodes we can call it a difficult problem. This difficulty can also be defined using the number of function and accumulators in the model.

If one thinks about the parameters, then they can realize that the difficulty parameters do not define how the schemas would be, but rather they define how difficult it is to apply those schemas in the problem after figuring them out. This is the rationale behind changing definitions of difficulty parameters. The updated DBN will solve various purposes, first it will define the difficulty of the problem and in no way difficulty of the schema, secondly it will be part of the emission probability features only, thus their usage while implementing the embedded BKT will be clearly defined which was not the case right now.

5.7 Summary

This chapter presented the results for both the algorithms. On one hand, the results for BNT-SM were not at all convincing, as they were effected by overfitting of the data, but FAST generated good results for emission features subskill model with AUC greater than 0.8. There was a promising improvement for the difficulty parameters, for the transition parameters model, but it moves towards a change in understanding of their definition. On comparing models, some very useful results were drawn, until now which were just assumptions. These results are bound to improve with availability of more data after trying these on varying population of students like for students from high-schools.

Chapter 6

EVALUATION WITH SIMULATED STUDENTS

As mentioned earlier, there are three common methods for evaluating student models. One is measuring the models fit to data, as was done in the preceding chapter. Another is to use the student model of a student to predict the students performance on a post-test (e.g., Anderson et al. (1990)). We could not do this, because a post-test would need several model-construction problems and thus would take too much time. We did include a one-problem post-test, but ultimately rejected its data as too unreliable as 47 out of 52 students had completed the test problem and the other 4 out of 5 had also created most of the nodes as well, with some errors. So, the post-test problem turned out to be relatively easy and was not able to classify the students correctly. The second method of evaluation was to define fit of the model using human data. ROC curves and AUC values were used to define the fit of the model. Results were presented in chapter 5. The third evaluation method is to use simulated students (Vanlehn and Niu, 2001). Each simulated student has partial knowledge, so it doesnt always answer correctly. The student model infers the simulated student's knowledge. The accuracy of its predictions against the actual knowledge of the simulated student comprises the evaluation.

6.1 Defining the Simulated Students

Our simulated students were created by defining the values of initial knowledge, learn, guess and slip. Since the assessment is embedded, the whole system was not required to run and just the simulated response was sent to the student module. The simulated response was calculated using $P(y_{q,t} = correct|\theta)$. Here θ, represents the model parameters for the simulated student. A random number was generated and if it was greater than this value then it was marked as incorrect and was marked correct otherwise.

The values were generated based on the kind of experiment we wanted to run, but there are some overall bound, presented in the literature, on the values of guess and slip (van De Sande, 2013)

$$P(G) + P(S) < 1 \tag{6.1}$$

where, $P(G)$ is the guess probability, P(S) is the slip probability. There was an extra check to make sure this bound was always true.

Simulated students ensures that we always know the unknown knowledge state of the student, as we know the mastery of a schema using the simulated student model. These values were used to test the reliability and validity of the assessment. Reliability means, given the same sequence of correct/incorrect solution and assessment will always assign the same level of mastery to the student. Validity means, assessment will represent a value close to the student's original value of learning. Reliability, is trivially true for a BKT as same parameter values will be multiplied for the same sequence of correct/incorrect responses. But to perform a better test of reliability, we changed this definition. It will be presented in the next section. Validity was really tricky to figure out, but again since we have the student model and we know how it is changing we can use it to check the validity of the assessment as well. The results for validity of assessment are presented in section 6.3.

Each simulated student was run for the same 10 problems which were used to learn the parameters in chapter 5.

6.2 Reliability

Testing reliability using the original definition, is trivial in case of a BKT. Due to the basic assumption of a DBN that is the parameter values do not change with time we can safely assume that BKT will always provide the same value for the assessment of a student. But we wanted to see if it is reliable when we use the same student, that is with same student model and same learn, guess and slip values, how far does the final assessment value varies. So, 100 students with 0.1 initial knowledge across all schemas, 0.2 guess, 0.1 slip and 0.05 learn value were created. At every step the correct or incorrect response was decided using the random number generator and the probability value calculated at each step. After that the knowledge of the student was updated assuming the response to be the actual response and student model in assessment module was also updated.

Four models of skill and subskills were used using transition constant features, which presented the best result in chapter 5. Table 6.1 shows mean and standard deviation of the final assessment value for exponential schema. Important point to note is the standard deviation as that shows how reliable the assessment was.

Skill model name	**Final Probability of Mastery**
Group1	0.871 ± 0.0286
Group3	0.853 ± 0.0178
Group4	0.868 ± 0.0252
Group6	0.817 ± 0.0524

Table 6.1: Final Probability of Mastery for Exponential Schemas Using Assessment Model, with Transition Constant Features.

6.3 Validity

Validity was hard to define with BKT models, since it depends on the students from which the parameters values were calculated. Thus we created simulated student data for the same 10 problems, and used FAST with transition constant model to find the parameter values. There were two sets of students created. Each set had 100 students (80 for training and 20 for testing using FAST). Then for each set of students, 20 new students were created as test data for the embedded assessment. Conditions created were student who had high initial knowledge (maximum of 0.75) and students who had low initial knowledge (maximum of 0.20) on some skills. The guess and slip were also picked randomly between 0.0 to 0.20 for guess and 0.00 to 0.10 for slip. Learn parameter was constant at 0.05.

This helped us perform two kind of analysis, first we could see how close were the value FAST parameters to the actual mean of the parameters that were used to create the data. Second was to calculate the total Root Mean square error for assessment over all the student. Which would tell us how different were the parameters from the actual student learning curve.

For exponential schema, the mean initial knowledge value was 0.39 $\pm$ 0.127 for first set of student and was 0.13 $\pm$ 0.085. The guess and slip parameters were around 0.12 and 0.05. Only first model was used to calculate the parameters for the data that is skills were defined as a combination of the schemas, and subskills were defined as schema_nodeType_property, that is exponential_accumulator_type (Group1). The parameter values calculated from the model were for set1 (high initial knowledge) learn parameter 0.064, and Initial knowledge of 0.362. For set2 (low initial knowledge) learn parameter was 0.079 and Initial knowledge was 0.162. The guess and slip parameters could not be compared as they had logistic regression values over the subskills. The total Root Mean Square Error across 20 new students was 0.0521 for high initial knowledge students and 0.0316 for low initial knowledge students. All these values are for exponential schema and the numbers are similar for other schemas as well.

In a class students vary somewhere in between very high knowledge to low knowledge. So we can safely assume that we have the upper and lower limits of the validity from the RMS error using two different set of students. The actual parameters were also calculated close to the mean value in both the cases. This was true for other schemas as well.

So, to summarize we were successfully able to test the reliability and validity of the assessment method and found it to be very convincing. This kind of analysis has been called the sensitivity analysis (Vanlehn and Niu, 2001) in literature, which gave us a better insight of the working of an assessment.

Chapter 7

CONCLUSION AND FUTURE WORK

This thesis undertook the problem of creating a student model for Dragoon, which is a new Intelligent Tutoring System for teaching modeling to students. There were multiple steps involved while working on this thesis problem. First, log data was gathered from students while interacting with Dragoon, then it was analyzed to understand what kind of knowledge tracing model would best suit to define the skills and subskills in Dragoon, and finally the student models were implemented and evaluated using two different modeling toolkits.

A total of 28 student models, based on 7 different ways skills and subskills could be combined in Dragoon. They were added to a Bayesian Knowledge Tracing (BKT), and the results were investigated:

- BNT-SM (7 models) modeled as logistic regression over subskills used as transition features. The results for this model were not that great as this was over fitting the data. The results for AUC value were close to 0.5 which usually means a random chance to decide whether student had mastered or not mastered the skill.

- FAST-constant-emissions (7 models) - this assumes that slip and guess parameters are same for all steps, all students and all times, but they can be different for different skills. This model showed better results than BNT-SM, but it showed that all the models of skill and subskills were similar. Even control model performed as well as other models which had no idea about the schemas.

- FAST-constant-transitions (7 models) - this assumes that learn and forget are the same for all steps, all students and all times, but can be different for different skills. This was the model which was the best fit for the human data. The results had an AUC values of around 0.83. It was also able to differentiate other models from control model which had an AUC of 0.813. It showed that the subskill which was defined as the complete node, was not a good way to understand the models and thus we can conclude that our current model of defining subskills as a property filling step in a node is correct. This was also used to infer a lot of understanding about the how students usually interact with Dragoon.

- FAST-constant-transition-with-difficulty - on top of FAST-constant-transitions model, difficulty parameters were added to see if could improve the already good results shown by the previous model (FAST-constant-transitions). This was not the case as it either performed at par with the constant-transitions models or it just didn't finish due to a very high increase in parameter numbers for skills. Only four out of the seven original models were parameterized using Dragoon.

This analysis of models helped us realize a lot of basic understanding like students do think about schemas as the control model was outperformed by constant-transition models. This was comforting to see as a lot of teaching in modeling using Dragoon was based on the schema which are the primary skills.

Simulated students were also created for the assessment models that we have proposed. It was found to be reliable and valid and the difference in numbers was primarily due to the randomness in the user response.

Going forward, first, we should look into the reasons as to why BNT-SM was overfitting the data and FAST wasn't. This would give us the clarity of checking our hypothesis by two different types of modeling techniques, even though they use expectation maximization to fit the data. The second thing one should look into is to construct models where number of parameters would decrease, and we can still have similar AUC accuracy. This would decrease the overhead of storage as well as the calculation due to higher number of parameters. One can also look at different ways of defining a model of skill and subskills and can improve the fit over the data. The new models can be defining subskills as skills and features as difficulty parameters. This would give us a separate value for each step and can help is many different comparisons.

The models proposed have increased the number of parameters used to define a student model by combining skills when a node belongs to two or more schemas. This can be a problem in future when we use schemas to decide next problems for students as combination can increase the total schemas that we want student to learn. For example, we have a knowledge value for linear and exponential. Then we have one for combination as well which exponential_linear. This new schema does not tell us if the student's learning for basic schemas has improved or not. There would $O(nm^2)$ combinations possible where n is the number of schemas and m is number of schemas to which a node can belong. So to teach all the schema combination would mean more problems with similar combination of schemas. The new approach would need some refining, to calibrate schemas from the combinations as well. For subjective probability model, to calculate the knowledge learning probability, we had a few assumptions like, equation, type, units, initial value, and description was the order in which weight values were assigned. Similarly, weights were assigned to nodes in the order of function, accumulator and parameter. These weight values can be calculated using Machine Learning techniques which will remove our judgement from it.

Time was not used for assessment or as a feature, but it has been realized to have a correlation with learning. This correlation can be harnessed in Dragoon and it might improve the AUC value for the model. In Dragoon, students get a hint once they provide a wrong solution twice. This does affect the student understanding of the modelling and if logs can be analyzed in a way to figure out the effect on Slip and Guess parameters, then even that can improve the Knowledge tracing values for Dragoon.

As mentioned earlier, Dragoon has several student modes that vary in the feedback that they give students. It also has an authoring mode where students create their own model instead of trying to create a hidden model. This thesis developed student models for only one mode: immediate feedback. It would be interesting to extend these student models to cover other modes as well. This would be especially difficulty and perhaps impossible for authoring mode, because we cannot define correct/incorrect response of a student. To do this one can change the features that define emission. For example, a student learns while checking how the values are changing using Graphs in Dragoon. This can be seen as a positive sign of learning. Time spent at where interaction points like nodes, graphs can be used as another feature.

REFERENCES

Anderson, J. R., Boyle, C. F., Corbett, A. T., and Lewis, M. W. (1990). Cognitive modeling and intelligent tutoring. *Artificial intelligence*, 42(1):7–49.

Chang, K.-M., Beck, J., Mostow, J., and Corbett, A. (2006). A bayes net toolkit for student modeling in intelligent tutoring systems. In *Intelligent Tutoring Systems*, pages 104–113. Springer.

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

d Baker, R. S., Corbett, A. T., and Aleven, V. (2008). More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing. In *Intelligent Tutoring Systems*, pages 406–415. Springer.

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

Khajah, M. M., Huang, Y., González-Brenes, J. P., Mozer, M. C., and Brusilovsky, P. (2014). Integrating knowledge tracing and item response theory: A tale of two frameworks. *Personalization Approaches in Learning Environments*, page 7.

Koedinger, K. R., Aleven, V., Heffernan, N., McLaren, B., and Hockenberry, M. (2004). Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In *Intelligent Tutoring Systems*, pages 162–174. Springer.

Marshall, S. P., Barthuli, K. E., Brewer, M. A., and Rose, F. E. (1989). Story problem solver: A schema-based system of instruction. Technical report, DTIC Document.

Mayer, R. E. (1981). Frequency norms and structural analysis of algebra story problems into families, categories, and templates. *Instructional Science*, 10(2):135–175.

Russell, S. and Norvig, P. (1995). Artificial intelligence: a modern approach.

van De Sande, B. (2013). Properties of the bayesian knowledge tracing model. *JEDM-Journal of Educational Data Mining*, 5(2):1–10.

VanLehn, K. (2006). The behavior of tutoring systems. *IJ Artificial Intelligence in Education*, 16(3):227–265.

VanLehn, K., Chung, G., Grover, S., Madni, A., Sande, B. v. d., Wetzel, J., and team, D. (submitted). Learning about dynamic systems and domain principles: The effectiveness of the dragoon intelligent tutoring system.

Vanlehn, K. and Niu, Z. (2001). Bayesian student modeling, user interfaces and feedback: A sensitivity analysis. *International Journal of Artificial Intelligence in Education*, 12(2):154–184.

Xu, Y. and Mostow, J. (2010). Using logistic regression to trace multiple sub-skills in a dynamic bayes net. In *Educational Data Mining 2011*.