TÉCNICO LISBOA

Student Model Learning

Francisco Manuel Franco Azeiteiro

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Manuel Fernando Cabido Peres Lopes

Examination Committee

Chairperson: Prof. Alberto Manuel Rodrigues da Silva
Supervisor: Prof. Manuel Fernando Cabido Peres Lopes
Member of the Committee: Prof. Claudia Martins Antunes

October 2017

Acknowledgments

I would like to thank my dissertation supervisor Prof. Manuel Lopes for his insight, support and sharing of knowledge that has made this Thesis possible.

This work was created as part of an investigation scholarship financed by the Fundação para a Ciência e a Tecnologia (FCT) and under the Seventh Framework Programme for Research and Technological Development (FP7) programme (FP7-ICT-2013-10610878).

Abstract

Intelligent Tutoring Systems (ITS) are computer systems used in learning environments in an attempt to optimize the students' learning experience. They provide performance improvements comparable to an expert teacher.

The student model in ITS consists of individually tracing the knowledge level of students that use it in terms of skills and/or sub-skills, with the goal of predicting whether or not a student is capable of solving any given exercise. The tutoring model then attempts to assert what exercise should next be proposed to the student in order to optimize his learning experience. Zone of Proximal Development and Empirical Success (ZPDES) and Right Activity at the Right Time (RiARiT) are tutoring model algorithms which are fairly independent of the underneath student model. This work shall ultimately attempt to have the ZPDES algorithm receive information for grouping students by estimating students' learning speed and proposing the model with approximately optimal parameters for students of that competence level. This was attempted by several different means including performing a diagnosis test before applying the algorithm itself, estimating the learning speed during the execution of activities and utilizing the optimal parameters for the most similar student profile. The obtained results show that this methodology allows for a better balance between the amount of skills learned and the number of time steps required to learn them than using the average parameters for every student, although the created model would need several adaptations in order to be used with real students.

Keywords

Student model; Tutoring model; Learning; Intelligent tutoring systems;

Resumo

Os ITS são sistemas computacionais utilizados em ambientes de aprendizagem numa tentativa de optimizar a experiência de aprendizagem dos alunos. Eles providenciam melhorias de desempenho comparáveis às de um professor.

O modelo de aluno num ITS consiste em traçar o nível de conhecimento dos alunos que o usam em termo de competências e/ou sub-competências, com o objetivo de entender que exercício deve ser proposto de seguida ao aluno de forma a optimizar a sua experiência de aprendizagem. O ZPDES e o RiARiT são algoritmos do modelo de ensino que são relativamente independentes do modelo de aluno subjacente. Este trabalho irá derradeiramente tentar fazer o algoritmo ZPDES receber informação para agrupar alunos, estimando a velocidade de aprendizagem destes e propondo o modelo com parâmetros aproximadamente óptimos para alunos desse nível de competência. Isto foi tentado de várias formas diferentes incluindo realizar um teste de diagnóstico antes de aplicar o algoritmo em si, estimar a velocidade de aprendizagem durante a execução de atividades e utilizar os parâmetros ótimos para o perfil de aluno mais semelhante. Os resultados obtidos mostram que esta metodologia permite obter um melhor equilíbrio entre a quantidade de competências aprendidas e o número de passos necessários para aprendê-las que o uso de parâmetros médios para todos os alunos, apesar do modelo criado precisar de várias adaptações para ser usado com alunos reais.

Palavras Chave

Modelo de aluno; Modelo de ensino; Aprendizagem; Sistemas de ensino inteligente;

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

BKT Bayesian Knowledge Tracing

HMM Hidden Markov Model

ITS Intelligent Tutoring Systems

FAST Feature Aware Student knowledge Tracing

RiARiT Right Activity at the Right Time

ZPDES Zone of Proximal Development and Empirical Success

ZPD Zone of Proximal Development

EM Expectation Maximization

AUC Area Under the ROC Curve

ROC Receiver Operating Characteristic

PFA Performance Factors Analysis

MAB Multi-armed Bandits

KC Knowledge Component

KTPC Knowledge Tracing that uses Partial Credit

Exp4 Exponential-weight algorithm for Exploration and Exploitation using Expert advice

IRT Item Response Theory

MDP Markov decision process

RNNs Recurrent Neural Networks

DKT Deep Knowledge Tracing

LSTM	Long Short Term Memory
KNN	K-Nearest Neighbors
RMSE	Root Mean Squared Error
LFA	Learning Factors Analysis
ARFF	Attribute Relation File Format
WEKA	Waikato Environment for Knowledge Analysis
GOZPDES	Group-Optimal Zone of Proximal Development and Empirical Success
FCT	Fundação para a Ciência e a Tecnologia
FP7	Seventh Framework Programme for Research and Technological Development

1

Introduction

Contents

1.1 Motivation

ITS are tutoring systems based on Artificial Intelligence programming techniques, providing some degree of individualized instruction [2]. Traditional versions of these systems can be broken down into several internal components [3], which handle in different layers the basic rules of the domain, tracing of the student knowledge, proposing the best possible exercise in terms of student learning, and the interactions between the user and the system.

Student model learning has been a popular area within ITS, its main goal is to accurately predict how a student will answer to a series of questions. Being able to achieve that goal would greatly increase the progress toward making ITS a powerful tool for teaching in general, allowing for more effective teaching systems to exist in several different mediums, from school education to advanced training in highly specialized environments.

Several ITS-based systems already exist and are used for different types of learning experiences in grade school environments [4] and otherwise, such as eTeacher [5] - a system that uses an intelligent agent to recommend courses to students in an e-learning system based on previous student performance in other courses, as well as the SHERLOCK system [6], used to train Air Force technicians in diagnosing electrical systems issues on F-15 jets, providing different levels of assistance depending on the performance of each trainee.

Several techniques in this area have been shown to have highly accurate prediction models in terms of the most used performance metrics in student model learning. Such models include the Performance Factors Analysis (PFA) [7] and Feature Aware Student knowledge Tracing (FAST) [8] algorithms. In fact, cheating models - models with some ability to read the student's future input or parameters - do not show a very large improvement in results, warning us that the quality of results in student model learning may be peaking [9].

1.2 Problem

All algorithms currently used in ITS either assume that all students are the same or that all students are different, and are therefore all treated equally or differently by the model depending on the case. The difference being that algorithms that treat students equally implicitly assume that every student possesses similar individual parameters by not modeling them and the only difference between students comes from their performance during exercises, while algorithms that treat students differently attempt to explicitly model the students' individual parameters, usually via estimation based on their performance during an exercise, or by giving these parameters to the students prior to solving exercises (this is usually done for testing purposes).

It is clear that students are not all equal. However, there are also certain limitations to making the

assumption that all students are different, in particular because the degree of similarity between them is not used in any way. How would a system that learns student models using past data be able to utilize this information in order to optimize the learning sequence of future students? At what point is the amount of gathered past data sufficient for it to be useful? Should only the data of students with similar performances to the one currently using the system be used for comparison? Can students be grouped up into types based on their performance in such a way that predicting a student's "type" can assist in the selection of the optimal learning sequence? Exactly how can the tutoring model algorithm make use of this additional information, and how should it receive it?

In particular, this work will focus on the ZPDES and RiARiT algorithms, which originally use little domain knowledge and are fairly independent of the algorithm being used for the student model. It would be interesting to analyze the outcome of utilizing one of these algorithms if supported by a system which estimates the students' attributes and proposes an optimal version of the previously mentioned algorithms for students of that competence level.

1.3 Hypothesis

The hypothesis being attempted will be the possibility of grouping students together based on certain criteria in such a way that the students are able to use models with optimized parameters for similarly competent students to their advantage, while still acknowledging that all students are different. Whenever a new student uses the system, his or her performance in certain exercises will allow the system to decide what group of students is estimated to have the most similar students in terms of learning capabilities. Challenges of this approach include accurately placing students on their correct group, making sure that the model parameters for each group of students are optimal, and deciding when grouping should start happening, as it will be difficult to accurately group up students if the overall number of students is low, since it is much more likely that a new student using the system will not properly fit into any of the existing groups. In the context of this work these issues can be resolved through either classification or clustering, depending on whether the analysis is performed on the group after the execution of the model or whenever a new student uses the system.

1.4 Objectives/Expected Contributions

The following are the expected contributions and objectives of this work:

- To obtain/explore the state of the art for student model learning techniques
- To learn and describe the ZPDES and RiARiT algorithms of the tutoring model

- To implement a version of the ZPDES algorithm supported by parameter estimation algorithms which allow the discovery of grouping criteria based on parameters of both the algorithms and the students. This new algorithm is called the GOZPDES algorithm
- To make the above system capable of grouping students based on their similarities under certain criteria, optimizing the learning sequence for each group as opposed to the typical approaches of optimizing it for each student or for a group containing every student
- To create a new student competence estimation method which requires few observations in order for it to provide improvements over treating all students equally in a time-constrained domain
- To create a new type of virtual students with the ability to learn skills at different paces based on their parameters
- To test the new implementation with the virtual students, and perform statistical analysis of the obtained results

2

Related Work

Contents

2.1 Background

This subsection shall contain the necessary information to have an understanding of the fundamentals of student model learning and ITS in general, as well as some simple algorithms and definitions which are needed to understand some of the more complex algorithms mentioned further into this document

2.1.1 ITS Architecture and Description

The typical architecture of an ITS contains the following models, existing in a layered form where each section is capable of communicating with the lower ones. The description of the responsibilities of each layer, from bottom to top, goes as follows:

Domain model (also called the cognitive model) - focuses on the problem-solving component, possibly containing rules, concepts and problem-solving steps or strategies [10]

Student model - focuses on tracking and predicting the progression of the student's knowledge level over several skills (and possibly sub-skills). Quality measurements for student models typically focus on how accurately the model predicts the correctness of the student's answers

Teaching model (also called the tutoring model) - attempts to propose to the student the best possible exercise to help the student learn at a given time, as well as providing the student with feedback on the answer. It utilizes information from the cognitive and student models to help it decide what type of exercise is more likely to help the student learn the most. Exercises for a particular skill typically stop being proposed once the student reaches a certain level of proficiency at it. [11]

Learning environment (also called the user interface model) - responsible for the user's direct interaction with the system, requiring knowledge of the possible interactions and their meaning, the ability to communicate its content, and any additional domain knowledge that may be required [12]

When combined, these modules compose a system which has a set of internal rules on how students function, is able to maintain information about each student's knowledge levels, proposes exercises based on the system's teaching goals and provides feedback on student's answers, potentially giving hints while promoting its ease of use. This set of characteristics is independent of the chosen ITS architecture.

2.1.2 Markov decision process

A Markov decision process (MDP) is a type of Markov model that may be used to mathematically model decision making problems while satisfying the Markov property (Markov, 1954), which states that

the probability of transitioning into a state only depends on the previous state, and consist of a tuple with the following parameters (for a thorough explanation see [13]):

- A set of states S
- A set of actions A
- Transition probabilities $P_a(s, s') = P(s_{t+1} = s'|s_t = s, a_t = a)$ representing the probability of transitioning from state s to s' at time $t + 1$ knowing that the action performed at time t was a
- A reward indicator $R_a(s, s')$ which states what is the reward for passing from state s to s' by performing action a
- A discount factor γ, which may be used to discount the value of future rewards

MDPs are a key part of several of the mentioned algorithms, any algorithm containing some form of reward function and probability based state transitions contains some form of MDP.

2.1.3 Hidden Markov Model

A Hidden Markov Model (HMM) is a type of Markov model in which the system respects the Markov property and the states are not directly observable. However, the output, which is state-dependent and based on emission probabilities, is observable [14]. A HMM is able to model this situation in such a manner that it is possible to make assumptions on what state the model is on at any given time based on the output, allowing estimations on future output and states to be made.

HMMs are used in the Bayesian Knowledge Tracing (BKT) algorithm, which is a very popular algorithm in student model learning and has generated several variants, making it important to have an understanding on what a HMM is in order to better understand the algorithm and its variants.

2.1.4 The forward-backward algorithm

When using a HMM the current state is not visible, put the observations are. The forward-backward algorithm calculates the probability of the observation sequence given the model [15] (this work by Rabiner was originally created for the domain of speech recognition).

Given the state transition distribution A, where the probability of transitioning from state i in time t to state j in time $t + 1$ is given by a_{ij}, the probability distribution of observing each possible output in state j, $B = b_j(k)$, the initial state distribution $\pi = \pi_i$, where π_i represents the probability of the initial state being i, the observation set $O = O_1, \ldots, O_T$, where O_t is the observation at time t, then a HMM can be given by $\lambda = (A, B, \pi)$, as well as the state sequence $Q = q_1, q_2, \ldots, q_t$, the forward-backward algorithm is no more than the product of the forward and backward algorithms after smoothing is applied. Both of

these algorithms can individually calculate the probability of an observation sequence. This algorithm is limited in that it can only find a local minimum, so convergence obtained from maximum likelihood estimations using the forward-backward algorithm are only able to reach a local maximum as well.

2.1.5 The forward algorithm

Considering the nomenclature used in the explanation of the forward-backward algorithm, the forward variable $\alpha_t(i)$ is defined as Equation 2.1 [16]:

$$\alpha_t(i) = P(O_1, O_2, \ldots, O_t, q_t = S_i|\lambda) \tag{2.1}$$

The inductive solution for $\alpha_t(i)$ is a three step procedure which works as follows:
Initialization step (Equation 2.2):

$$\alpha_1(i) = \pi_i b_i(O_1) \tag{2.2}$$

Induction step (Equation 2.3):

$$\alpha_{t+1}(j) = [\sum_{i=t}^{N} \alpha_t(i) a_{ij}] b_j(O_{t+1}) \tag{2.3}$$

Termination step (Equation 2.4):

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.4}$$

Where N is the number of states.

2.1.6 The backward algorithm

Considering the nomenclature used in the explanation of the forward-backward algorithm, the backward variable $\beta_t(i)$ is defined as Equation 2.5 [16]:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_T, q_t = S_i|\lambda) \tag{2.5}$$

The inductive solution for $\beta_t(i)$ is a two step procedure which works as follows:
Initialization step (Equation 2.6):

$$\beta_T(i) = 1 \tag{2.6}$$

Induction step (Equation 2.7):

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \tag{2.7}$$

Where $t = T-1, T-2, \ldots, 1$. t moves backwards since this algorithm starts by computing the probability of the final observations.

2.1.7 The Expectation Maximization algorithm

The Expectation Maximization (EM) algorithm attempts to iteratively calculate the maximum likelihood of a set of parameters. It is split into two steps: The E step, which finds the distribution for the unobserved variables which, in the ITS domain, often refers to the student's knowledge state of skills and, when applicable, sub-skills. This distribution is found through the current estimate of the parameters, as wells as the values of the observed variables. The M step consists on re-estimating the parameters to be those with maximum likelihood, assuming that the distribution found in the E step is correct [17] [18]. There are several variants of this algorithm depending on the domain. When working with HMMs, the Baum-Welch algorithm can be used to estimate the model's transition, emission and initial probabilities [15]. This algorithm will be used to estimate student attributes when required.

2.1.8 The Baum-Welch algorithm

Given the nomenclature previously defined in the forward-backward algorithm and $\xi_t(i,j)$ representing the probability of, given the observation sequence and the model, being in State S_i at time t and S_j at time $t+1$, which can also be represented as Equation 2.8:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \tag{2.8}$$

Which given the definitions of $\alpha_t(i)$ and $\beta_t(i)$, can be rewritten as Equation 2.9:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=t}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1} \beta_{t+1}(j))} \tag{2.9}$$

It is now possible to define $\gamma_t(i)$, the probability of, given the observation sequence and the model, being in state S_i at time t based on $\xi_t(i,j)$ with Equation 2.10 [16]:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{2.10}$$

And since by definition $\sum_{t=1}^{T-1} \gamma_t(i)$ can be used to obtain the expected number of transitions from S_i and $\sum_{t=1}^{T-1} \xi_t(i,j)$ is the expected number of transitions from state S_i to state S_j, it is now possible to

reestimate the parameters A, B and π, since γ and ξ can be used to obtain estimates of these values, the exact formulas are shown in Equations 2.11, 2.12 and 2.13 [16]:

$$\bar{\pi}_i = \gamma_1(i) \tag{2.11}$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.12}$$

$$\bar{b}_j(k) = \frac{(\sum_{t=1}^{T} \gamma_t(j)|O_t = v_k)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.13}$$

Where the model itself can be reestimated given a sequence of states Q through Baum's auxiliary function [16] (Equation 2.14):

$$Q(\lambda, \bar{\lambda}) = \sum_{Q} P(Q|O, \lambda) \log[P(O, Q|\bar{\lambda})] \tag{2.14}$$

Which can be repeated until convergence.

2.1.9 ROC and AUC

The Receiver Operating Characteristic (ROC) curve is a graphical plot that plots the true positive rate (the rate of individuals with a particular trait which were classified as having the trait) in function of the false positive rate (the rate of individuals without a particular trait which were classified as having the trait). Typically the $x = y$ line, which represents random chance, is plotted in the middle. Any value underneath this line does not provide an improvement over random selection.

Area Under the ROC Curve (AUC) measures the area underneath the ROC curve, its significance can be seen as the probability that a randomly selected positive instance will be ranked higher than a randomly selected negative instance. In student model learning, the value of AUC can be used as a measurement of how accurately an algorithm can predict the student's knowledge state. An algorithm with an AUC of no more than 0.5 is no better than random selection, and is therefore not useful.

2.1.10 Bayesian Knowledge Tracing

BKT [19] has been a popular approach to the problem presented by ITS in the student model, and some of the more modern approaches are based on this work. The original work consists on having students trying to learn a set of skills related to Lisp, Prolog or Pascal programming by asking them to solve exercises which require the use of those skills in a practice environment, the system also attempts to infer if the student has learned any one skill by using a HMM [16] with two states for each one:

the learned state, where the student knows the skill, and the unlearned state, where he doesn't. It is assumed that once a student learns a certain skill it cannot be forgotten, meaning that there is a possible transition from the unlearned state to the learned state, but not the other way around.

The model possesses four parameters:

- $P(L_0)$ is the probability that the student possesses a particular skill prior to the first opportunity to apply it.
- $P(G)$ is the probability that the student will answer correctly to the question if the skill is in the unlearned state (guess).
- $P(S)$ is the probability that the student will answer incorrectly if the skill is in the learned state (slip).
- $P(T)$ is the probability that the skill being learned transitions from the unlearned to the learned state after an opportunity to apply it, independently of the student getting the answer correct or incorrect.

Corbett and Anderson attempt to calculate how likely it is that a student understands a particular subject matter using the following formula:

$$P(L_n) = P(L_{n-1}|evidence) + (1 - P(L_{n-1}|evidence)) \times P(T) \tag{2.15}$$

The probability predicted for the student to answer the next question correctly depends only on the results of the previous exercise testing the same skill.

Equation (2.15) can be broken down into Equations (2.16) and (2.17), where the $evidence$ is represented as $O_n = \{o_1, \ldots, o_n\}$, which represents all answers until time step n, and o_n can be either correct or incorrect [20]:

$$P(L_n|O_n) = 1 - \frac{(1 - P(T))[1 - P(L_{n-1}|O_{n-1})]P(G)}{P(G) + (1 - P(S) - P(G))P(L_{n-1}|O_{n-1})}, o_n = correct \tag{2.16}$$

$$P(L_n|O_n) = 1 - \frac{(1 - P(T))[1 - P(L_{n-1}|O_{n-1})](1 - P(G))}{1 - P(G) - (1 - P(S) - P(G))P(L_{n-1}|O_{n-1})}, o_n = incorrect \tag{2.17}$$

Figure 2.1 shows the result of a simulation made in Java based on Equation (2.15) for a single skill, the y-axis represents the system's predicted probability of the skill being in the learned state given how the student has answered until then, the x-axis represents whether the student gets the answer wrong (W) or correct (C), for example, "4W4C3W" means that the student got 4 answers wrong, then 4 answers correct, then 3 answers wrong. The parameter values used were $P(L_0) = 0.1, P(G) = 0.05, P(S) = 0.05$ and $P(T) = 0.3$

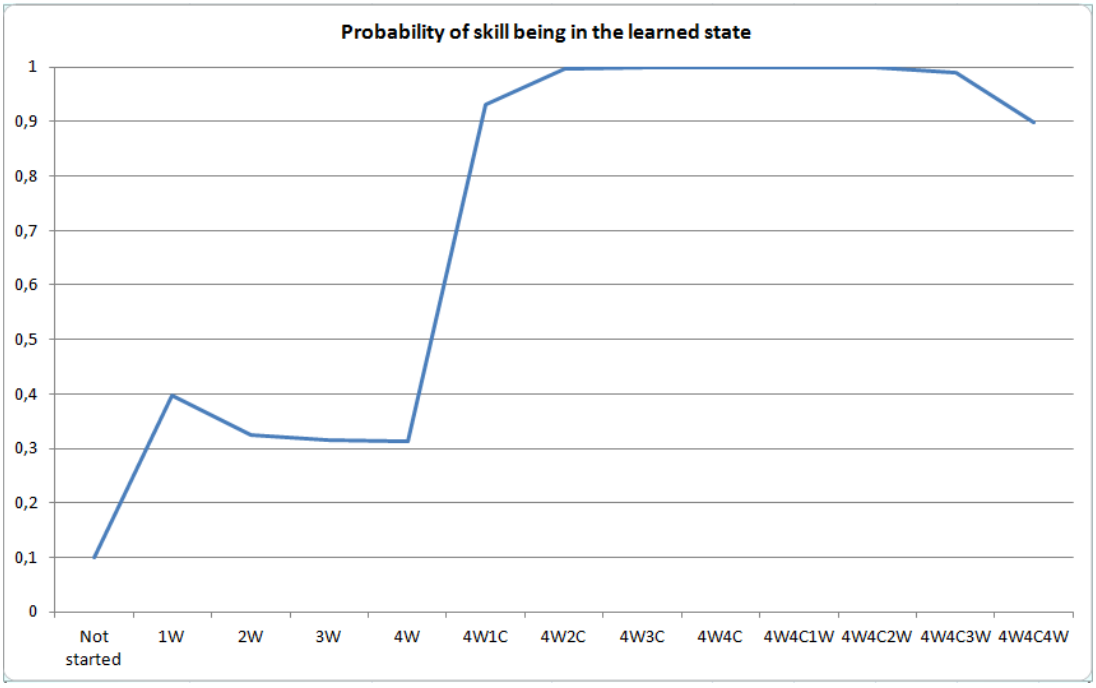

Figure 2.1: How the system's prediction of a skill being in the learned state changes according to the student's answers.

It was simulated that the student got 4 answers wrong in a row, followed by 4 correct answers, and finally 4 wrong answers. Figure 2.1 shows that, even though the student does not forget a skill once it is learned, it is possible for the system to decrease its estimation of the probability of the student answering the next question correctly. With the lowest possible probability that can be estimated by the system after the student attempts to answer at least one question being 0.3, as even a student who gets every answer wrong has a probability of $P(T)$ to learn the necessary skill. Also, after answering a few questions correctly in a row the system is almost certain that the student has learned the skill (meaning that the skill is in the learned state).

There are many algorithms either based on or related to BKT, such as:

- Bayesian Evaluation and Assessment model - a model focused on assessing the impact of help in student performance. [21]
- Individually estimating $P(L_0)$, $P(G)$, $P(S)$ and $P(T)$ for each student based on the EM algorithm. [22]
- Add the concept of difficulty by changing the probability of slip ($P(S)$) or guess ($P(G)$) for each exercise, as opposed to for each skill. [23]
- Sub-skills tracing by using Logistic Regression [24]

The original BKT algorithm has the issues of representing the understanding that students have of skills as either learned or not learned, such binary interpretation is not always true, there are also the issues that it is quite difficult for any one exercise to have a clear, well defined and accurately measured set of associated skills and that each exercise can only use one skill. An expert in knowledge representation is often required to handle these issues.

The BKT algorithm is important to this work because it is a simple, yet popular algorithm in the area of student model learning, having generated several variants, and because the implemented solution's concepts of students and exercises use some of the BKT parameters.

2.1.11 Multi-armed Bandits

In the original Multi-armed Bandits (MAB) problem [25], a gambler has a series of K slot machines to play, being able to pull the arm of one of them at each time step t, which provides a certain reward. The distribution of the reward provided by each slot machine is fixed, but unknown. The gambler's task is to choose which slot machine to play. This task is not simple, as the gambler must worry about the exploration versus exploitation problem: which machine provides the highest average reward? How quickly can the gambler find the best slot machine in terms of average reward? How many times must the gambler test each machine before figuring out if that machine is worth exploiting or not? The gambler's performance can typically be measured by the "regret" - the difference between the expected return of the optimal strategy and the gambler's expected return [26].

The optimal solution and order of the expected regret for the MAB problem is dependent on the domain - in the above environment the rewards provided by the slot machines are independent of each other, but if the domain switches to the teaching environment described in the ZPDES and RiARiT section, then the assumption that the learning activities are independent of each other cannot be made - several activities are of the same type of exercise, sometimes of varying difficulty, and even if the exercise type is different it is possible that the same Knowledge Component (KC)s are being used in different proportions. The proposed solution was to consider the situation as an adversarial bandit problem, where the only assumption made is that each learning activity (or slot machine considering the original gambler problem) is assigned an arbitrary and unknown sequence of rewards, one having been assigned to each time step, which is chosen from a bounded real interval [26]. For the implementation of the ZPDES and RiARiT algorithms the adversarial bandit problem was resolved by using an adaptation of the Exponential-weight algorithm for Exploration and Exploitation using Expert advice (Exp4) algorithm [26], which also allows for expert advice to be taken into consideration.

2.1.12 Supervised and Unsupervised learning

Supervised learning refers to the task of utilizing labeled training data (usually called a training set) in order to infer a function. Using the training set, the goal is for the algorithm to correctly classify the data within a testing set. Supervised learning can be divided into two different tasks depending on the nature of the data:

- Regression - used when dealing with continuous values for the data (for example, a person's weight in kilograms)
- Classification - used when dealing with discrete values (for example, a person's gender)

In unsupervised learning the data is not labeled, and the goal is to learn the data's structure. Due to the lack of labels, it is difficult to measure how accurately an unsupervised learning algorithm is.

Regarding the work being done, there is no clear labeling of how the students should group up, and the parameter used to determine each student's learning capability is a value that needs to be estimated each time. It therefore fits into the unsupervised learning category. After groups have already been established, there will then be a task of placing new students into these groups. This constitutes a classification problem in supervised learning, as the groups constitute discrete values.

As such, this work will put some focus on supervised and unsupervised learning algorithms which may be useful to solve this work's grouping problems. There are two especially important fields to mention here: cluster analysis - which focuses on establishing groups, and statistical classification - which focuses on assigning new observations to previously existing groups.

2.1.13 Cluster analysis

Cluster analysis consists in finding groups in data in such a way that the items in a given group are more similar to each other than to items from outside of the group [27]. Multiple clustering algorithms exist, and the most adequate one for each case, as well as its internal parameters, are dependent on the dataset. The process of cluster analysis can be broken down into four steps [28]:

1. Feature selection or extraction (also called data representation) - selecting what features to use in the clustering algorithm. Good features are typically the ones with clear distinction patterns and those which are easy to extract and interpret. This is a difficult and important challenge, as proper data representation allows for even the simplest clustering algorithms to find the desired clusters [29].
2. Clustering algorithm design or selection - choosing, adapting or creating the clustering algorithm which works best with the given data and features, as well as what parameters to give to the

algorithm when applicable. Criteria such as the number of data points and features selected (dimensionality) are important to take into account when choosing what clustering algorithm to use, as there is no one algorithm in this area that is always better than all others.

3. Cluster validation - the ”procedures that evaluate the results of cluster analysis in a quantitative and objective fashion” [30]. There are three different categories of testing criteria: external indices, internal indices, and relative indices, each using external (using a priori information), internal (based on the data itself), and relative (other types of quantitative comparison) criteria respectively. These criteria are defined on three types of clustering structures: partitional clustering, hierarchical clustering, and individual clusters [30].

4. Results interpretation - the process of obtaining knowledge from the way that the clustering has occurred. Experts of the relevant field(s) are typically needed to ensure that this process is done properly and that the obtained result is useful.

Examples of algorithms in this area include the K-means algorithm and fuzzy clustering, which does not necessarily force each object/item into a single cluster, and instead claims that every item may have a certain value of ”belonging” to more than one cluster, with higher values for more similar clusters.

2.1.14 Statistical classification

Within supervised learning, the task of statistical classification (sometimes just called classification) is to, given a new observation and a set of categories, decide to which category (or categories) the new observation belongs. In the domain of this work, the classification problem will arise when groups of students have already been made and a new student appears, leading to the issue of deciding which previously existing group this new student belongs to.

2.1.15 K-nearest neighbors

The K-Nearest Neighbors (KNN) algorithm consists in the classification of data points into a certain class depending on the class of the k nearest neighbors [31]. Let $R = r_1, r_2, \ldots, r_m$ be a set of m reference points and $Q = q_1, q_2, \ldots, q_n$ be a set of n query points in dimensional space d, the KNN algorithm classifies each query point by giving it the same class as the winner of a majority vote of its k nearest neighbors. Alternative classification methods based on this algorithm include weighting each neighbor differently depending on the distance to the query point [32] and the 1-nearest neighbor classifier, which works similarly to KNN with $k = 1$.

When there is a tie in the number of nearest neighbors for multiple different classes the most common solution is to run the KNN algorithm again for $k = k - 1$, which is eventually guaranteed a solution

since $k = 1$ is guaranteed to not have a tie. The most common method of search for KNN is a brute force algorithm, which computes the distance between all query points and reference points and selects the k closest for each query point. The distance can be measured in several different ways, such as Euclidean or Manhattan distance, as long as the same measurement technique is used for all distance measurements.

2.1.16 Other work

Work in Psychology and Social Science has shown that motivation is an important part of learning, and that students can draw motivation from performing exercises or solving problems of an ideal difficulty in such a way that the challenge provided is on par with the student's competence levels in the area. Motivated students working on challenging exercises tend to have better performance grade-wise than equivalent unmotivated students [33].

2.2 Limits to Accuracy

In an attempt to determine what the upper bound of accuracy is on student modeling, predictive models [9] that simulate being able to perfectly predict certain key moments in student learning were created, with limitations that still allow for each model to make incorrect predictions, as a predictive model that is always correct does not provide useful information in terms of the upper bound that other models can achieve. Three different variants were made, each with a different level of "perfect" predictions:

- A model that knows if the student will answer the next question correctly and assumes that the student has learned the skill in that case. This model is forced to make an error in its prediction if the student gets a question wrong after the system assumes that it has learned the skill, after which it is allowed to decide if it wants to change its prediction of the student's knowledge state.
- A version of the previous model that is capable of understanding when the student has learned a skill, but unable to see the correctness of the student's next question before the answer affects the accuracy of the model. This variant behaves similarly to the first one, with the exception that it is less capable of predicting the first correct answer that the student has for each skill.
- Evaluating the student knowledge of each skill as a value between 0 and 1 for each skill, while doing the same for item difficulty, combining this approach with the prediction of the next answer from the first model. This variant makes an incorrect prediction when the student answers incorrectly to an exercise which the system predicted was of lower difficulty than the student's skill, the student's estimated knowledge for that skill can then be lowered.

The results were obtained for two different datasets: ASSISTments data - a web-based tutor aiming to teach math to eighth-grade for a total of 343 students, and the 2010 KDD Cup dataset (an annual Data Mining and Knowledge Discovery competition) - containing 607,026 rows of data from 574 students. The most successful variant in terms of both AUC and R^2, a normalized variant of Root Mean Squared Error (RMSE) was the third variant, with the concepts of item difficulty and non-binary student knowledge. In comparison to the Deep Knowledge Tracing (DKT) algorithms, which used a different section of the ASSISTments data, the value for the AUC is only slightly higher.

2.3 State of the Art

2.3.1 The PFA algorithm

One of the main limitations of BKT is that it is not capable of modeling exercises to have multiple KCs. However, it is clear that many exercises and real world problems would require the use of multiple KCs to solve properly. This means that BKT is limited in the type of exercises that it can use. The Learning Factors Analysis (LFA) [34] model has been used within the area of educational data mining, and has the ability to handle student learning through three parameters: The subject's ability (a parameter of the subject), the easiness and the learning rate (parameters of the KC). Despite the ability to handle exercises with multiple KCs, the LFA model is not a proper student model algorithm, as it ignores the students' response to each question, and therefore does not differentiate between correct and incorrect answers.

PFA is an adaptation of LFA which attempts to take advantage of its ability to manage multiple KCs in an exercise, while also making it sensitive to student performance, which indicates student learning. Its formula can be seen in Equation (2.18).

$$m(i, j \in KCs, s, f) = \sum_{j \in KCs} (\beta_j + \gamma_j s_{i,j} + \rho_j f_{i,j}) \tag{2.18}$$

Where m is a representation of accumulated learning for student i, j is a KC, β is the easiness of a KC, s tracks the student's previous success in the KC, f tracks the student's previous failures in the KC, with γ and ρ being used to scale the effects of their respective observations.

The parameters are fit in order to maximize the log-likelihood of the model for 4 different datasets, these datasets include teaching several different Math exercises for grades 5 to 12, as well as grades 9 to 12 Physics. The results were measured in a series of different terms, including log-likelihood and AUC. The results showed that PFA performs generally better than BKT, while still providing fairly equivalently to LFA outside of the student model learning domain.

2.3.2 The FAST algorithm

The FAST algorithm [8] offers an approach to the student model that takes advantage of data that can be obtained from e-learning systems, allowing for the representation of multiple sub-skills supported by expert knowledge. Logistic regressions are used instead of conditional probability tables, slip and learning probabilities, making FAST scale linearly with the number of parameters, as opposed to most algorithms of the BKT family's exponential scaling. FAST uses the EM [35] with Features algorithm [36]. A variation of the EM algorithm in which the E step remains similar, but the values of the parameters for the M step are calculated with a logistic function and in function of weights β and features $f(t)$, where in the case of the FAST algorithm f is a feature extraction function which uses the observations to create the feature vector $f(t)$ and β is obtained by training a weighted regularized logistic regression. The logistic regression is trained by weighting each observation to the probability of that observation having been obtained from the latent states. There are three possible types of features, which can be active at different times: features active prior to the student mastering the skill, features active after the student masters the skill, and features which are always active.

FAST's ability to add parameters to itself means that it can be modeled into a more efficient equivalent to models of the BKT family, it can also make and update different guess (when the student hasn't learned the skill but gets the answer correct) and slip (when the student has learned the skill but gets the answer wrong) probabilities for each skill or sub-skill, since these can be modeled independently.

Regarding item difficulty, most algorithms of the BKT family make the assumption that all items for practicing a skill are of the same difficulty [37], while Item Response Theory (IRT), a classical psychometric paradigm, allows different difficulties for each item, but does not allow student learning due to its static nature, leading to it underestimating item complexity because the harder items are usually done at a time when the students have already been learning, leading the system to believe that the item is easier than it actually is due to not taking student learning into consideration. FAST models IRT using feature engineering through logistic regression with binary variables indicators [8]. Finally, FAST may also utilize features indicating items, as well as if the student answers an item template correctly or not. The FAST model featuring both of these characteristics shows better results in terms of AUC than those with only one or none of these features, as well as the PFA and original BKT algorithms.

2.3.3 Deep Knowledge Tracing

The main issue in student model learning is the knowledge tracing task of modeling the student's knowledge level as the student completes exercises. Correct knowledge tracing allows the system to be able to foresee how a student will perform in a related exercise based on previous efforts. An accurate knowledge tracing algorithm is most likely complex because human learning is a complex task to model.

DKT attempts to tackle this issue by applying Recurrent Neural Networks (RNNs) [38] to knowledge tracing by allowing the existence of latent knowledge states using artificial neurons which allow the representation of student knowledge to be learned from data.

The original BKT algorithm has the issues of unrealistic representation of the student's knowledge state and required skills are often ambiguous and require expert knowledge to be properly modeled.

RNNs allow for hidden neurons to evolve based on previous activation and system input, this makes the spreading of information throughout the neural network a recursive process, meaning that previous information is never truly lost, and may influence a decision/prediction at a later point.

DKT is implemented using two different types of neural networks:

- Traditional RNNs which use a series of hidden states, each state h_t with the exception of the first is influenced by an input vector x_t and the previous hidden state h_{t-1}, and produces an output vector y_t. Equations (2.19) and (2.20) show the value of each hidden state and output vector:

$$h_t = \tanh(\underline{W}_{hx} x_t + \underline{W}_{hh} h_{t-1} + b_h) \tag{2.19}$$

$$y_t = \sigma(\underline{W}_{yh} h_t + b_y) \tag{2.20}$$

  Where $\underline{W}_{hx}$ is an input weight matrix, $\underline{W}_{hh}$ is a recurrent weight matrix, $\underline{W}_{yh}$ is a readout weight matrix, σ is a sigmoid function, and b_h and b_y are latent and readout units biases respectively.

- A variant of RNNs known as Long Short Term Memory (LSTM), where latent units only lose their values if cleared by a "forget gate", allowing for information to be maintained more easily. LSTM can perform more complex calculations for the same number of latent units due to each hidden unit's value being updated through multiple interactions.

The obtained results show an improvement of 11 to 24% in AUC comparing to BKT depending on the dataset, used datasets include virtual students answering the same sequences of exercises, a benchmark dataset, and Khan Academy - a popular educational platform with a total of 47,495 students performing a total of 1.4 million exercises.

2.3.4 Machine Teaching

Machine Teaching [39] provides an alternative paradigm to ITS and an inverse problem to machine learning - while machine learning attempts to create computer programs with the ability to learn and discover information by themselves, machine teaching attempts to specifically teach a machine learning algorithm a target model θ^* by providing it with a training set D. An optimal training set can be seen as the one with the smallest value of $|D|$.

Machine teaching can also be applied to humans with the goal of designing a personalized optimal lesson for each student. The main difference between the approaches of ITS and machine teaching is that ITS algorithms typically assume students to be a black-box function, where the goal is to discover the optimal input sequence that will generate the best results in terms of learning, while machine teaching explicitly seeks to create a cognitive model of the student in order to compute the student's optimal training set. One possible way of doing this is by attempting to create cognitive models and conducting experiments to see what each cognitive model's optimal training set is. The task of creating an optimal cognitive model for each student is obviously very difficult, so making a series of cognitive models which accurately cover most types of students would be a good alternative.

Machine teaching is still a new area of research, and will surely face several problems such as how to accurately create cognitive models for each student, and properly establishing its theoretical foundations. However, it has the potential of making ITS much cheaper to produce due to not explicitly needing a student or tutoring model thanks to the depth of its student cognitive model. If the issues of the area are resolved, a new architecture of ITS may take advantage of machine teaching in an attempt to produce cheaper and simpler systems with comparable results.

2.4 Directly Influential Work

2.4.1 The simplified Exp4 algorithm

The Exp4 algorithm [26] is a MAB algorithm which aims to use expert advise to resolve the adversarial bandit problem as optimally as possible. Considering the existence of N different experts, and the choice between K possible actions, the advice vectors $\xi^i(t)$ to $\xi^N(t)$ contain the weight that each expert gives towards choosing each action. The probability of performing a particular action is calculated based on the sum of the weights that the action has from every advice vector relative to the total weight, as well as an exploration rate γ. The selected action is then performed and a reward r is obtained. This reward value is then used to update the weights of every advice vector in such a way that the higher the obtained reward was, the more important the advice vectors that gave a high weight to the selected action become.

Exp4 has some issues when being adapted into the teaching domain for use in the ZPDES and RiARiT algorithms, particularly due to the possible prerequisites which exist between activities, as well as the fact that the weight of an activity can never be decreased due to the value range of r being between 0 and 1. This can be problematic, as the activities to be weighted are only those in the ZPD, as well as the issue that a poor learning activity that has been in the ZPD for some time is very likely to have a greater weight than a new activity which may be considerably better for the student to learn skills. In order to adapt Exp4 to this domain the used formula is slightly changed: for an activity a its recent

rewards w_a is updated with the formula $w_a \leftarrow \beta w_a + \eta r$, where r is the reward obtained by performing activity a, which may be negative. β and η determine the confidence in the new reward, with $\beta + \eta = 1$. The probability of choosing to perform activity a is given by Equation (2.21), where ξ_u is an advice vector equivalent to uniform distribution, $\tilde{w}_a$ are normalized w_a values ensuring that the probability distribution is valid and γ is the exploration rate, meaning that higher values of γ increase the uniformity of the probability distribution.

$$p_i = \tilde{w}_a(1-\gamma) + \gamma\xi_u \tag{2.21}$$

2.4.2 The ZPDES and RiARiT algorithms

In ITS, the tutoring model focuses on choosing the activities that it believes will be better for improving the student's competence levels. In the case of the ZPDES and RiARiT algorithms little information from the cognitive and student models is used, leading to a weak dependency on these. In fact, these algorithms attempt to be independent of predefined cognitive and student models, and estimate the characteristics of individual students online while attempting to provide each student with a motivating learning experience. The exercises originally proposed consisted on seven KC proposed to 7-8 year old students, and had components such as the capability to add and subtract integers or decomposing decimal numbers, this was made in a context of handling money in order to buy or sell items [1].

An adaptation of the Exp4 algorithm is used in order to associate weights to each activity. These are used to assist in calculating the probability that each activity in the ZPD has of being selected.

The ZPDES algorithm, inspired by the ZPD [40] and the empirical estimation of learning progress [41], requires very little knowledge about the problem. The learning progress r is computed using the following equation where C_k is 1 if the exercise at time k was solved correctly, and 0 otherwise:

$$r = \sum_{k=t-d/2}^{t} \frac{C_k}{d/2} - \sum_{k=t-d}^{t-d/2} \frac{C_k}{d-d/2} \tag{2.22}$$

Equation (2.22) calculates the difference in amount of correct answers of the last $d/2$ samples and the previous $d/2$ samples. This shows whether the student's success rate for a particular activity is increasing or not. A value of $r \leq 0$ indicates that the student is not learning the required KCs for the activity. Before a reward for an activity can be computed that activity must have been attempted a minimum of d times. There will typically be too many activities to explore for this method to be effective, therefore an expert is expected to define the ZPD as a graph with prerequisites between activities, as well as connections between activities without prerequisites between them. An activity which has others as prerequisites will be inserted into the ZPD when the success rate of its prerequisites is equal to or greater than its expansion threshold and those activities have a minimum of d attempts, while

ZPD expansion into activities with no prerequisites can happen when an activity connected to them has achieved a value of r lesser than or equal to 0. This allows separation of activities in subsets, where the subset of the activity can indicate the difficulty of the exercises within the subset based on a required competence level for each KC, allowing the expert to be able to determine whether or not the student is able to progress into a particular subset based on the student's performance on an easier subset of the same activity. Additionally, an activity cannot be removed from the ZPD if some activities which were never put into the ZPD have it as a prerequisite. This prevents situations where a chain of activities is not performed because one of its initial activities was removed too quickly.

Additionally, when a sequence of increasingly difficult activities is being performed, it is possible for the weight of the easier ones to be decreased when the harder versions are being answered correctly. To be specific, for a sequence of activities $a1 < a2 < \ldots$ where a_i is an easier activity than a_{i+1}, if $w_{a_i} < \theta w_{a_{i+1}}$ and $\sum_{k=1}^{t} \frac{C_k(j)}{t} > \omega$, then $w_{a_i} = 0$ and $w_{a_{i+I}} = w_{a_{i+I-1}}$, $\omega < 1, \theta$ and I are parameters that determine how sensitive the ZPD is to these cases.

Activities can be broken down into vectors such that Activity $a = \alpha_1, \ldots, \alpha_m$, where each α is either 1 if the KC can be learned from it or 0 otherwise, allowing for any Activity to be able to teach any number of skills. It is possible for KCs to influence each other, so $\beta_{i,j}$ represents the impact that KC K_j has on the probability to learn K_i, and $\beta_{i,i}$ represents the probability to learn K_i independently from other KCs [42]. The probability of learning a KC at each activity step is given by Equation (2.23). Essentially the probability of knowing KC K_i at time step n is the probability of learning the skill independently of other skills plus the sum of the impacts that every skill that is already learned in $n-1$ has on learning K_i, as long as $\alpha_i = 1$.

$$p(K_i^{(n)} = 1 | L^{(n-1)}, a^{(n)}) = \alpha_i(\beta_{i,i} + \sum_{j \neq i}^{m} \beta_{i,j} K_j^{(n-1)}) \tag{2.23}$$

It is possible to allow exploration between unrelated activities not necessarily of higher or lower difficulty. Figure 2.2 shows the evolution of the ZPD for a particular student. Initially an expert defines the exploration graph, with each activity possibly having a pre-condition before the activity may be proposed. The student initially gets activity A1 correct, updating the ZPD. After the system realizes that the student is not progressing in activity A2, the ZPD is updated to not include A3, extending instead to a different activity type in C1.

The RiARiT algorithm requires more information of the domain as well as the student in an attempt to explicitly estimate each student's level of competence for each activity and update it according to the student's performance in the activity itself. Each activity can potentially require the usage of several KCs in different measures. The student's competence level in a given KC is modeled as a continuous number between 0 and 1, where 0 indicates that the KC has not been acquired at all, and 0.9 indicates that the KC has been 90% acquired. A student's estimated competence level for knowledge component KC_i is

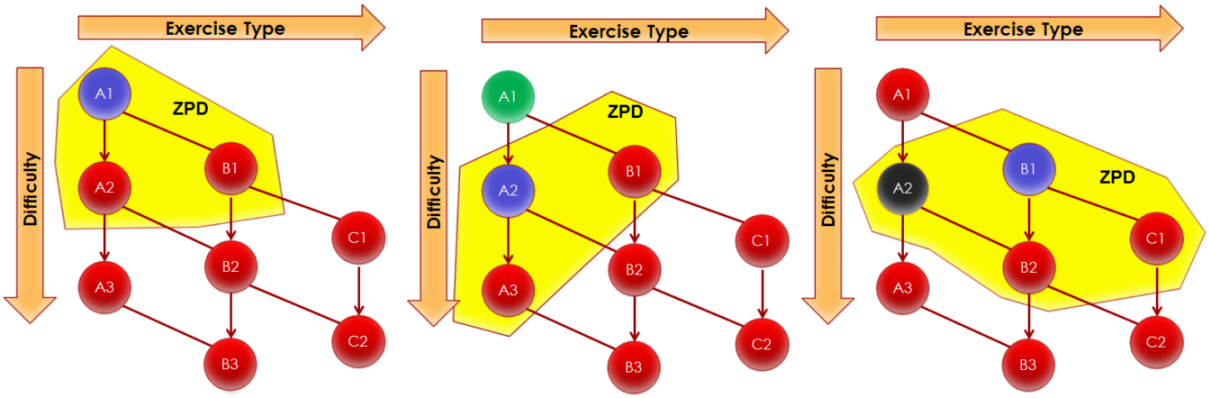

Figure 2.2: From [1], example of the progression of the ZPD for a student that solves activity A1 correctly, then fails to progress in activity A2, being asked to resolve activity B1 afterwards and expanding the ZPD to include an unrelated activity C1.

given by c_i, and for activity a the competence level required in KC_i to fully succeed in said activity is given by $q_i(a)$. Whenever an activity is performed the reward is computed by the following equation:

$$r_i = q_i(a) - c_i \tag{2.24}$$

Equation (2.24) only applies in cases where the student succeeds in the activity and $q_i(a) > c_i$ or the student fails in the activity and $q_i(a) < c_i$, corresponding to the cases where the student is expected to have a lower competence level than required to succeed in the activity but succeeds, and the case where the student is expected to have a higher competence level than required to succeed in the activity but fails. For all other cases $r_i = 0$ because they provide little information.

The reward calculated is then used to update the student's estimated competence level according to Equation (2.25):

$$c_i = c_i + \alpha r_i \tag{2.25}$$

Where α is an adjustable parameter that corresponds to the confidence in each new piece of information.

The results include an evaluation comparing the effectiveness of the algorithms with virtual students and actual user studies with 400 students aged 7 to 8 years old. The tests with virtual students include two types of students: Q Students, which are able to use all activities to learn, albeit at different learning rates and maximum comprehension levels and P Students, where each student may not be able to perform certain activities at all, this student type attempts to simulate students with problems or disabilities that may be an issue in education environments such as illiteracy (unable to understand written exercises) or deafness (unable to understand oral exercises)

The published results indicate that students tend to succeed at higher levels of several different KCs than in a sequence defined by the teacher, with the exception of the first type of exercise. This happens because the expert sequence spends a much larger amount of time with this type of exercises, while the ZPDES and RiARiT algorithms propose different types of exercises much sooner. In addition, the linear sequence of exercises proposed by the expert sequence has trouble dealing with students with difficulty in the KCs required for the initial exercises, while the ZPDES and RiARiT defined sequences tend to move to different types of exercises if the student is unable to progress in a particular type.

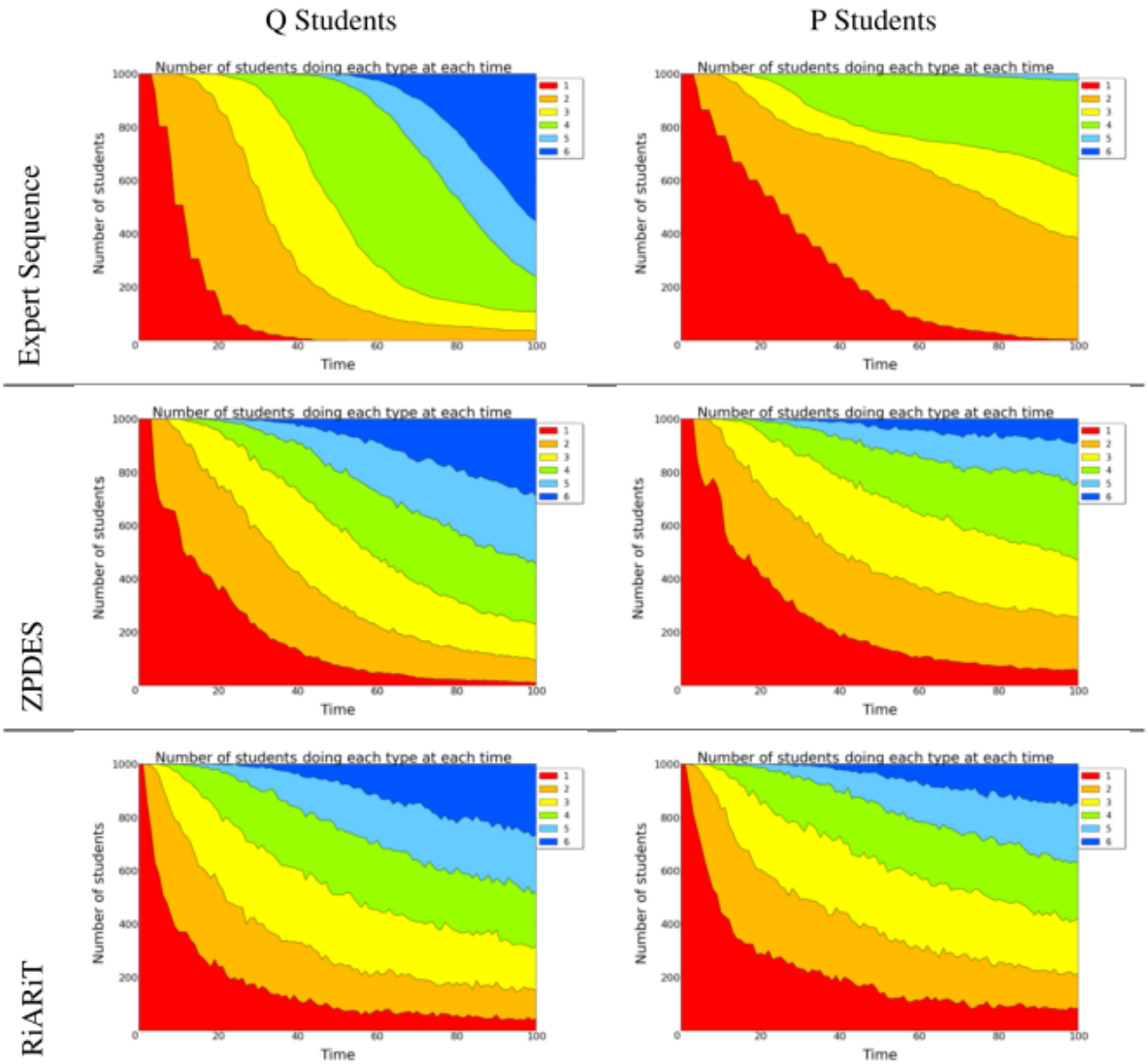

Figure 2.3: From [1], comparison of the expert sequence, ZPDES and RiARiT algorithms in terms of the number of students doing a certain exercise difficulty over time for virtual students. ZPDES and RiARiT propose more difficult exercises earlier, and keep proposing easier exercises longer if necessary.

2.4.3 The KTPC algorithm

Another variant of BKT, and an algorithm that RiARiT inspires its reward updating equation from. In the Knowledge Tracing that uses Partial Credit (KTPC) algorithm [43] the guess and slip probabilities are obtained from two Gaussian distributions with a given mean and standard deviation for each one. Partial credit can be assigned as the score is given as a value between 0 and 1, where 1 is the score given if the student answers correctly without asking for hints. Every time that the student asks for a hint the potential right answer score gets discounted by $\frac{1}{\#hintsavailable}$, meaning that if there are 5 hints possible and the student asks for two hints before answering correctly the given score is 0.6. In case of an incorrect answer a penalty is assigned which depends on the type of question: for "fill the blank" type questions a wrong answer provides a discount of 0.1, for multiple choice questions a wrong answer provides a penalty equal to $\frac{1}{\#wronganswers}$, for example: getting a true or false question wrong warrants a penalty of 1. The total discount provided is the sum of discounts for hints requested and penalties for wrong answer to a minimum of zero. Certain questions are also scaffold questions, meaning that if the student's score in the question is less than 1, then the student is given a series of scaffold questions, guiding the student step by step through the solution. In these cases, the score given to the student is 90% of the average score of the scaffold questions.

2.4.4 Hint Factory

Hint Factory [44] attempts to automatically provide students with hints which are in line with the problem solving strategy of the student. The environment where the system is placed has several possible actions to perform from any one state, and typically several of them are correct. The system uses knowledge from how previous students have solved the problem and models each student's possible states and actions as a MDP, where the reward function contains a small negative reward for every non-solution state. This causes the optimal policy to be the one where the student reaches a solution state the fastest. Whenever the student requests a hint the system provides the hint which should lead to the successor state with the highest expected reward.

Hint Factory was originally designed to improve student performance on a previously existing tutorial in the domain of logic proofs. It is possible to adapt this into ITS by having an expert initially predict frequent correct and incorrect approaches, as well as create appropriate hints for each situation. Due to the nature of the Hint Factory system, it is possible to implement expert knowledge into it in the form of solutions while additionally having automatically generated hints.

Hint Factory's importance to this work comes from the exemplary usage of MDPs and from the fact that it is a system which attempts to improve the learning experience of students based on the past performances of other students.

2.4.5 K-means

The k-means algorithm is a popular clustering algorithm that attempts to divide the data set into k clusters of equal variance, in such a way as to minimize the sum of the squared error over all clusters (the sum of distances between each data point in a cluster and the center of the cluster where that data point belonged) [29]. The algorithm works as follows:

1. Select k random means
2. Create k clusters by associating each data point with its nearest mean
3. The mean of each cluster is recalculated to be the centroid of each cluster based on the mean between data points
4. Repeat steps 2 and 3 until the algorithm reaches convergence

Variations of the algorithm include the k-medians clustering, which recalculates the centroid of the cluster with the median of each cluster instead of the mean [45].

3

Solution Proposal

Contents

3.1 Code Rundown

This section contains a rundown of the initial code, some classes will be omitted due to not being directly related to the work which was done afterwards.

The base architecture that was used is publicly available on `https://github.com/flowersteam/kidlearn_lib`. It contains the original RiARiT and ZPDES algorithms, programmed in Python, and requires having the numpy, scipy, seaborn, matplotlib, paramiko, uuid and importlib packages installed.

Initially, the students are created based on certain configuration - it is possible to use students with fixed characteristics and fake KCs for demonstration purposes, or generate a given number of students with varying initial knowledge levels of each skill based on normal distributions with fluctuating values of mean and variance within a certain interval. Q students have a guaranteed learning capability, which ensures that they are able to learn all KCs, albeit at differing paces depending on the student. P students may have certain limitations which prevents them from performing certain exercises. There are other limitations that make these virtual students an imperfect representation of real students, such as the fact that motivation is very difficult to properly model and how it is somewhat unrealistic to propose certain types of exercises to P students to begin with - if a P student is unable to see, then in reality an exercise that required vision would either never have been proposed to the student in the first place, or the student would receive special assistance such as having the teacher read the exercise.

A working session is then created for the students by associating them with a sequence manager (ZPDES for this work), which is responsible for choosing the next activity at each time step. Each time step executes the following procedures in order:

1. The sequence manager chooses what activity the student will do next;

2. The student answers the activity

3. The activity result is saved

4. The sequence manager updates its estimates (depends on the algorithm used)

According to Clement, each activity has a series of parameters determining the difficulty level, the type of exercise, the contents of an exercise and the presentation of its content. An activity is a set of all possible exercises which can be defined with a set of parameters.

The test experiments with the virtual students typically have 100 time steps. It is possible to ask the system for its estimate of the student's knowledge levels for each KC in the previous time step, as well as generate graphs based on the results.

3.2 Discussion

This section will recall how the related work and fundamentals have helped building and solidifying the approach that will be presented in the solution proposal.

As a popular student model algorithm, BKT serves as the foundation for several other algorithms, and is used as a baseline for the comparison of results in those, as well as other student model algorithms. In particular, the probabilities of guess and slip are modeled in some way by almost all algorithms in the area. An example of an alternative to the BKT architecture is the FAST algorithm, which models features using the EM with Features algorithm. Its architecture allows the use of a varying number of features, and FAST can perform similarly to BKT, as well as some BKT variants, depending on the number of features used and what they are used on. Its main impact on the work is the use of the EM with Features algorithm, as one of the approaches that will be attempted is the use of a variant of the EM algorithm as a criterion to assist in the clustering of the students.

The ZPDES algorithm will be used as the basis of the tutoring model for the proposed solution. This algorithm originally attempts to implement MAB's solutions to the exploration versus exploitation problem into ITS in an attempt to optimize the student's learning experience by determining what the best next exercise is to perform in terms of having the right difficulty for the student. The KTPC algorithm inspires RiARiT's reward updating equation, which the proposed solution shall partly implement. The goal of the work will be to provide the ZPDES algorithm with information from the experience of past students in such a way that whenever a student is using the system, he or she will be inserted into a group of students with similar characteristics, and will therefore run a model with parameters which have proven to be the optimal one for students of those characteristics. The main difficulty of this work will be in discovering these characteristics and the algorithms that will be implemented to assist in this. The Hint Factory system is relevant to this work because it also attempts to improve the student's learning experience based on the past performances of other students.

3.3 Approach

This work will focus on attempting to adapt the ZPDES algorithm in order for it to be able to utilize the results generated by the student model learning algorithm, which will attempt to place the students into groups based on each student's performance and provide a model with optimal parameters for each group instead of attempting to estimate it for each individual student or using the same ones for every student. This implies the use of several other algorithms in order to estimate, group and compare the students.

Whenever a new student is using the system, he or she will be placed into one of those groups based on an estimation obtained from the performance had in certain exercises or within a certain

amount of time steps. By establishing a different set of model parameters for each group based on its members' learning speed it is possible that such a system may yield better results than algorithms which consider all students to be equal or different, but do not take advantage of the similarities between them. The reason for this is because such a system can potentially identify the group that a new student belongs to much faster than an algorithm that considers all students to be different would adapt to the student's individual characteristics. The use of MAB in the ZPDES and RiARiT algorithms may assist in this endeavor thanks to the ability that MAB have to swiftly resolve the exploration versus exploitation problem, potentially allowing for a fairly fast but accurate identification of what group each student should be placed on.

3.4 Architecture

3.4.1 Clustering

Like it has been said before, the main challenges of this work are regarding the grouping aspect - in particular what criteria to use to group the students and how to group them having said criteria. A variant of the EM algorithm will be used to attempt to extract features which may be used as criteria for the system. Other techniques for criteria selection will be manual and take into consideration the exercises proposed. For example, if the system is made to offer the same initial exercises for every student, it is possible to use the value of r from the ZPDES algorithm after a fixed amount of exercises (Equation (2.22)). The effectiveness of these criteria must be evaluated, as it is very difficult to predict theoretically whether they will work or not.

3.4.2 Classification

The classification task in this work consists in deciding what group each new student belongs to. The learning speed is used as the criterion used to distinguish among different student groups, meaning that the most important part in this step is to accurately estimate this value, which is programmed as an attribute of each student. The estimation performed cannot directly obtain this attribute, as this would be impossible to do when interacting with real students, so the Baum-Welch algorithm will be used instead. Additionally, the created groups are based on student profiles, having one at the center of each group.

3.4.3 Parameter Optimization

Each group of students will have a set of parameters which determine how the GOZPDES algorithm will function for each group. Every student within a certain group will use the same GOZPDES parameters. Within each group the parameters are chosen as the optimal values that minimize the average

number of time steps for that group's student profile while maintaining the probability of learning all skills above a certain threshold. Since the parameters are optimal for the profile, then as long as the classification step is done correctly all students will have approximately optimal parameters, since the most optimal parameters for each student of the ones available are the ones which are optimal for the profile student with the closest learning speed attribute value.

4

Implementation

Contents

This chapter will focus on describing the general architecture of the program, as well as shortly explaining the classes and libraries used.

4.1 Architectural overview

The original Python program with the ZPDES and RiARiT algorithms is available, as previously mentioned, on `https://github.com/flowersteam/kidlearn_lib`, with the developed solution of this work being developed in Java based on the previously existing functionalities. The reasons for changing the programming language used are for allowing the reconstruction of the code, both as a learning process and as a way to modify the underlying structure of the classes in order to allow a new architecture with simpler, easier to understand classes and methods, as well as allowing the use of already existing Java libraries. In particular, the Baum-Welch algorithm - the used implementation of this algorithm was obtained from the jahmm library [46], which also contains a Java implementation of HMM.

The implemented solution - called the GOZPDES algorithm - contains adapted versions of the BKT, Exp4 and ZPDES algorithms.

4.2 Class rundown

Students of different capabilities are implemented through two attributes called $baselearn$ and $baseinit$, with $baselearn$ influencing the probability that the student has of learning skills and $baseinit$ influencing the probability of the student already knowing each skill before solving an exercise containing said skill.

Students are modeled differently from the typical BKT formula - knowledge of each skill can still be seen as an HMM with two states: not knowing and knowing the skill. There is a possible transition from not knowing to knowing the skill, but it is not made in the same way as originally done in Corbett and Anderson's work. Given a skill sk and a student st, the probability that the student has of learning the skill after each exercise, assuming that the student didn't know the skill before, is given by the following formula (Equation (4.1)):

$$P(T_{st}) = baselearn_{st} \times x + P(T_{sk}) \times y \tag{4.1}$$

Where $baselearn_{st}$ is the student's $baselearn$ attribute, $P(T_{sk})$ is skill sk's $P(T)$ value, and x and y are two factors that influence the weight of $baselearn_{st}$ and $P(T_{sk})$, with $x + y = 1$.

A similar equation was used to determine the student's probability of knowing the skill before attempting to apply it in any exercise (Equation (4.2)):

$$P(L_{0st}) = baseinit_{st} \times z + P(L_{sk}) \times t \tag{4.2}$$

Exercises in this work contain a list of skills, with each one having a certain weight measured from 0 to 1 which determines how important that skill is in the exercise. A larger weight on a skill means that the exercise component of that skill is larger - the skill has a larger influence in successfully completing the exercise, and is therefore also a better opportunity for the student to learn the skill than an exercise in which the same skill has a lower relative weight. As such, the weight of a skill in an exercise is used in estimating the student's knowledge at the skill, as well as in determining the probability that the student actually has of answering the exercise correctly.

Each skill contains similar attributes to a typical exercise in BKT - with probabilities of guess ($P(G)$), slip ($P(S)$), transition ($P(T)$) and initial learning ($P(L_0)$). Since each exercise may use multiple skills with different weights it is possible for different exercises to use the same skills without being equivalent to each other.

Upon entering the ZPDES domain the concept of exercises is replaced with activities, which work similarly in terms of skills and influences, but also allow for the existence of relationships between them in a tree structure, making it possible for certain activities to only be added to the ZPD if certain conditions in their parent or branch activities are met.

4.3 Test students, skills and exercises

This section describes the attributes and/or parameters of the students, skills and exercises which are used for testing purposes unless specifically mentioned otherwise.

4.3.1 Students

As mentioned previously, this work's implementation of students has two attributes: $baselearn$ - which affects the learning speed, and $baseinit$ - which affects the probability of knowing a skill before trying to learn it in an exercise.

The used test students have the following $baseinit$ and $baselearn$ values respectively:

- Student1: 0.0, 0.1
- Student2: 0.1, 0.8
- Student3: 0.1, 0.65
- Student4: 0.0, 0.2
- Student5: 0.0, 0.05
- Student6: 0.0, 0.1

- Student7: 0.0, 0.15
- Student8: 0.05, 0.2
- Student9: 0.05, 0.2
- Student10: 0.05, 0.1

4.3.2 Skills

This work's implementation of skills is in a way similar to how it is applied in BKT, with each having the same set of probabilities. The main difference is that the $P(T)$ and $P(L_0)$ values are only one of the factors that determine the real probabilities of transition and initial knowledge, these attributes contribute to the real probabilities according to the formulas of Equations (4.1) and (4.2).

The skills used for testing purposes have the following $P(L_0)$, $P(T)$, $P(G)$ and $P(S)$ values in that order:

- Skill1: 0.1, 0.3, 0.05, 0.05
- Skill2: 0.1, 0.2, 0.1, 0.1
- Skill3: 0.2, 0.5, 0.03, 0.03
- Skill4: 0.05, 0.15, 0.05, 0.05
- Skill5: 0.05, 0.1, 0.05, 0.05
- Skill6: 0.05, 0.5, 0.05, 0.05
- Skill7: 0.05, 0.25, 0.05, 0.05
- Skill8: 0.05, 0.2, 0.05, 0.05

4.3.3 Exercises

While the skill implementation of this work is by itself very similar to BKT, the same is not the case for exercises. Unlike in BKT, each exercise may use multiple skills and each skill may have different weight in the exercise. The exercises used for testing purposes use the following skills, along with the respective weight:

- Exercise1: Skills 1 through 7, weight 0.3 for each one
- Exercise2: Skill2 and Skill7:

  - Skill2 weight: 0.8
  - Skill7 weight: 0.4
- Exercise3: Skill4
  - Skill4 weight: 1.0
- Exercise4: Skill1 and Skill3:
  - Skill1 weight: 0.4
  - Skill3 weight: 0.6
- Exercise5: Skill5 and Skill6:
  - Skill5 weight: 0.5
  - Skill6 weight: 0.5

4.4 GOZPDES implementation and parameter analysis

This section studies this work's version of the ZPDES algorithm and its implementation, from explaining its use of MAB to how the optimal parameters were found.

4.4.1 Analysis of Expansion threshold

It is important to discuss what values to use as thresholds to expand the ZPD and to remove activities from it - too high a value would force the students to solve exercises within the same activity several times after already learning the relevant skills. At the same time, if the expansion threshold is too low the probability of the student completing an exercise through lucky guesses increases, which causes the algorithm to falsely assume that the student has learned the required skills. For example, assuming an activity a which only uses 1 skill, if the threshold to remove an activity is 0.8 and a student gets the first two attempts wrong, after which he learns the skill, the student must then correctly answer the activity 8 times in a row before that activity can be removed. For each time that the student answers the activity incorrectly, he must then answer correctly 4 times to make up for it in order to reach the required threshold. This may lead to a waste of both time for a real student and computational resources when using virtual students. A test was therefore made to understand how many answers it takes for different kinds of students to complete a sequence of activities at different ZPD expansion thresholds. This will also show if the number of answers has any potential as a criterion to distinguish between different types of students. For this test a linear sequence of activities was made and repeated 20.000 times, with only 1 activity ever being in the ZPD. Once that activity's success rate is equal to or greater than

Table 4.1: Average number of steps to finish going through the sequence of activities for students 1, 2, 3 and 4 while changing the threshold to expand the ZPD.

Expand threshold	Student1 time	Student2 time	Student3 time	Student4 time
0.10	46.0	26.8	28.3	39.3
0.20	58.0	28.3	30.3	46.9
0.30	76.4	32.3	35.3	60.0
0.40	105.7	38.9	43.7	80.3
0.50	130.4	44.3	50.5	97.9
0.60	173.4	56.1	64.4	128.9
0.65	202.6	64.8	74.0	149.7
0.70	245.1	77.1	89.3	182.5
0.75	303.0	91.4	106.4	224.0
0.80	402.0	120.2	140.2	294.5

Table 4.2: Average probability of knowing every skill after going through the sequence of activities for students 1, 2, 3 and 4 while changing the threshold to expand the ZPD.

Expand threshold	Student1 % skills	Student2 % skills	Student3 % skills	Student4 % skills
0.10	5.6	49.4	41.0	10.6
0.20	12.3	54.1	45.7	18.9
0.30	27.6	64.1	57.3	34.6
0.40	64.1	84.9	81.7	68.1
0.50	75.8	89.1	87.2	79.0
0.60	87.1	94.3	92.9	88.4
0.65	89.8	95.2	94.3	90.7
0.70	97.8	99.1	99.1	98.1
0.75	98.3	99.2	99.2	98,5
0.80	99.4	99.7	99.7	99.4

the expansion threshold, then the activity is removed from the ZPD and the next activity is added in. The order of activities contained Exercises 2 through 7 in that order, followed by Exercise 1. All activities had to be performed a minimum of 3 times before they could be removed from the ZPD. Tables 4.1 and 4.2 show the results for Students 1, 2, 3 and 4. Students 2 and 3 have considerably higher $baselearn$ values than Students 1 and 4. The $baselearn$ values for Students 1, 2, 3 and 4 are 0.1, 0.8, 0.65 and 0.2 respectively.

Before analyzing the tables it is important to mention that this was still an incomplete and simplistic version of the GOZPDES algorithm - MAB are not used and there is only ever one activity in the ZPD. This will allow to examine what the ideal value for the expansion threshold should be, but not much else.

Regarding the results seen in Tables 4.1 and 4.2, it is clear that the students with lower $baselearn$ values require considerably more steps to finish every proposed activity, but given a high enough expansion threshold there is not a noticeable divergence in the probability of knowing every skill by the end. As for the ideal value for the expansion threshold, it seems that values between 0.7 and 0.8 provide the best probability of the students knowing every skill by the end of the algorithm's execution, while at the same time not requiring too many time steps to finish. Setting this threshold to 0.7 at this point seems to

offer a good balance between the number of steps to finish and the probability of the students knowing every skill at the end. It should be clear that the goal of any future parameter variation will be to achieve the best possible balance between a high probability of the students knowing all skills and a number of total steps as low as possible, as in a real world environment having the students learn the skills is the main goal of this work, but that mustn't be done in an excessive amount of time if systems of this type are to be made viable for use in time constrained environments like, for example, schools.

The values obtained for expansion thresholds above 0.8 were not shown because Students 1 and 4 took far too many steps to finish without presenting a significant gain in the probability of knowing every skill.

4.4.2 Skill difficulty

Representing the concept of difficulty with the knowledge that the student has being state-based is not simple, as the progression of what the student knows for each skill cannot be shown as an intermediate value, and exercises using some of the same skills are not necessarily independent of each other. A way to represent the increase in difficulty of an exercise type is by working around the concepts of exercises and skills: an Exercise e_i is considered to be more difficult than an Exercise e_j if e_i requires every skill required by e_j plus at least one additional skill, regardless of every skill's parameters and weight in the exercise. This idea can be transfered into activities, since these are essentially treated as exercises placed within a hierarchy. This way, a student who ends up progressing through an activity due to a sequence of lucky guesses will once again be stuck in the next, more difficult activity until the student has hopefully learned the required skills. This will, however, likely have one problem: since the probability of a student getting an exercise (or activity) correct is given by the geometric mean based on the probabilities of guess and slip for each skill, it is very much possible that a student correctly answers an exercise for which he does not know at least one skill. This will happen more frequently for exercises with many skill requirements, so long as the student knows every previous skill.

4.4.2.A Impact of skill difficulty in student performance

To confirm the above hypothesis a similar test to the previous one was made, using the same sequence of exercises as the one used to obtain Tables 4.2 and 4.1, except that this time every activity additionally uses the skills required by every activity before it at a weight of 0.5. The results are shown in Tables 4.3 and 4.4, the maximum success threshold tested was 0.85 this time because the changes in the concept of difficulty has made it so that students require less time steps to make it through all of the activities.

Analysis of Tables 4.3 and 4.4 shows that adding the concept of difficulty causes the probability of the students knowing every skill by the end of all activities to decrease, especially for students with lower

Table 4.3: Average number of steps to finish going through the sequence of activities for students 1, 2, 3 and 4 while changing the threshold to expand the ZPD after adding the concept of difficulty.

Expand threshold	Student1 time	Student2 time	Student3 time	Student4 time
0.10	34.9	23.1	23.9	30.2
0.20	39.8	23.5	24.4	33.1
0.30	48.2	25.1	26.5	38,9
0.40	62.4	29.0	31.1	49,3
0.50	76.1	31.5	34.5	58.6
0.60	102.1	38.4	42.9	77.8
0.65	122.0	43.7	49.3	92.0
0.70	154.6	53.4	60.5	115.5
0.75	190.6	62.3	71.4	142.6
0.80	262.7	81.8	94.3	192.0
0.85	397.5	121.2	140.3	293.5

Table 4.4: Average probability of knowing every skill after going through the sequence of activities for students 1, 2, 3 and 4 while changing the threshold to expand the ZPD after adding the concept of difficulty.

Expand threshold	Student1 % skills	Student2 % skills	Student3 % skills	Student4 % skills
0.10	11.2	62.4	54.8	19.9
0.20	12.9	63.3	54.4	21.2
0.30	16.9	63.4	56.2	24.5
0.40	23.5	69.3	62.0	32.7
0.50	29.3	71.3	65.1	37.3
0.60	41.5	76.8	71.8	48.9
0.65	49.9	79.9	75.5	56.7
0.70	67.9	89.4	86.3	72.8
0.75	74.1	90.8	88.4	77.2
0.80	85.6	94.6	93.7	87.4
0.85	92.3	97.2	96.5	93.8

$baselearn$ values. The average time that the students are taking has also decreased.

Due to the way that this model is made, every time that a student tries to answer an activity (after every time step) there is an opportunity to learn every skill of that activity, but since the average time spent doing activities decreases they are less likely to learn the required skill. This is due to the way that the probability that the student has of correctly answering an exercise is calculated, as mentioned previously. This means that the more skills an activity uses, the more likely it is for the student to go through the activity without learning the new skill because the previously learned skills are actually making it likely that the student answers correctly anyway. This is not necessarily a poor representation of the real world, as exercises with many skills only exist in this model as more difficult versions of other exercises. An argument can be made that it is more likely for a student to correctly answer a difficult exercise of a type that he is accustomed to than he is to answer a relatively simple exercise of a type that is unknown to the student. Manually checking the knowledge state of each skill shows that the skill that is by far the most likely for the students to not know is Skill7 - the skill that is only used for the last activity.

Given that the probability of the students knowing every skill and number of steps to finish has gone down, updating the threshold value to around 0.8 seems to now offer the best balance between these two. Adding the concept of difficulty to the code makes the tutoring model considerably more powerful, since it makes exercises much more realistic.

4.4.3 Implementation of Multi-armed Bandits

For implementing the MAB component, a simplistic version of the Exp4 algorithm [26] was used. Originally a series of advice vectors assign weights to each possible activity. The activity is chosen based on probabilities obtained based on the weight of each activity in function of the total weight of all activities. The chosen activity is then performed and the advice vectors are updated based on the obtained reward and on the probability of the activity having been chosen. If the reward is high, then the advise vectors that gave that activity greater weight are considered as better than those that gave that activity a lower weight. For a detailed explanation of this algorithm's adaptation for this domain please check section 2.4.1.

Given that the reward is calculated similarly for GOZPDES as it is for ZPDES (Equation (2.22)), it is possible for the weight assigned for each activity to be negative. However, in order to ensure that the probability obtained in Equation (2.21) has a minimum value of $\gamma\xi_u$, corresponding to a random selection based on the exploration rate, the minimum value that can possibly be assigned to an activity's weight is set to 0. An alternative solution would have been to not allow a negative reward value, but the chosen approach has the advantage that negative rewards continue to have a greater impact in decreasing an activity's weight. The argument that could be made in favor of forcing the minimum reward value for

Equation (2.22) to be zero is that given the model's state-based approach to student knowledge, it is not possible for a student to unlearn a skill, meaning that a negative reward should not be more influential than a reward of zero. However, the minimum weight approach is likely to be a better fit when using real students instead of virtual ones because their knowledge is not necessarily state-based, despite it being more likely to be influenced by student slips than the minimum reward approach. Due to the total weight not being able to ever be zero because of the need to regularize the weights, the minimum weight is set to be the lowest possible positive value of a double in Java, which is 2^{-1074}.

4.4.4 Test model

Having properly implemented every feature of GOZPDES, a new set of exercises were turned into activities which were placed into a structure usable by the ZPD. Figure 4.1 shows the activity hierarchy used for the next tests. An arrow indicates a prerequisite relationship - A2 is only added to the ZPD when the success rate in A1 passes the expansion threshold. Connections without arrows indicate activities without prerequisites between them which will be proposed when the leftmost activity gets a reward value lesser than or equal to zero - if the reward obtained for performing A3 is ever zero or less then the ZPD expands to include B2, but the opposite cannot happen.

In addition to the previously mentioned features of ZPDES, a method that automatically removes activities if they have a "child" activity of one of their own "child" activities in the ZPD has been implemented. For example, if activities A1 and A3 with the relationship showed in Figure 4.1 are in the ZPD, then A1 will be automatically removed from it. This method complements the previously existing feature that would sometimes allow for a new activity to be added to the ZPD, and provides a small decrease to the number of time steps taken to go through the ZPD without changing the probability that the students have of knowing all skills.

The exercises used to create activities A1, A2, A3, B1, B2, B3, C1 and C2 are all new, while using the same skills as before and with every skill having a weight of 0.8. As for the skills used in each activity, the used skills are added in numerical order, with activities using the same letter and a greater number being more difficult, following the previously defined concept of difficulty. The list of skills used for each activity is the following:

- A1: Skill1
- A2: Skill1, Skill2
- A3: Skill1, Skill2, Skill3
- B1: Skill4
- B2: Skill4, Skill5

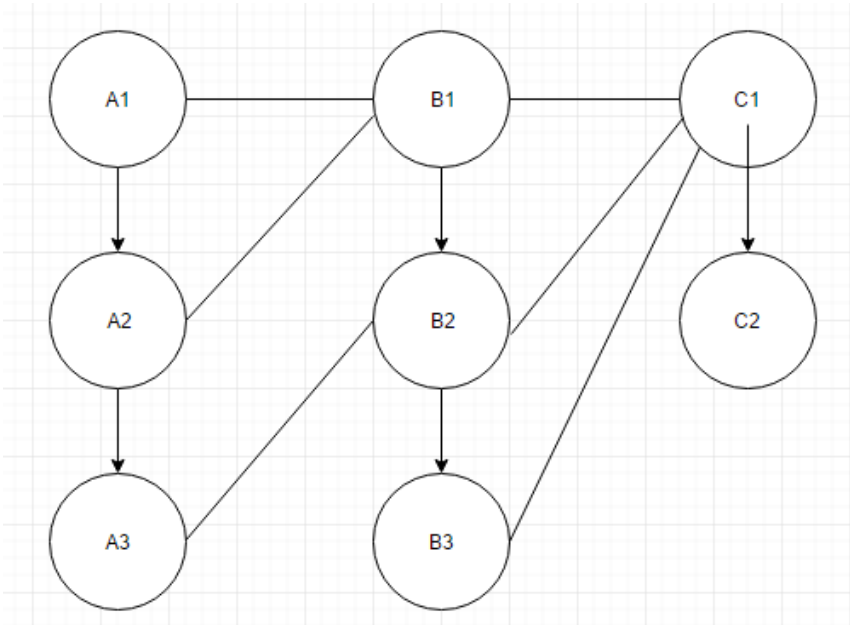

Figure 4.1: Relationships between a series of different activities used for testing.

- B3: Skill4, Skill5, Skill6
- C1: Skill7
- C2: Skill7, Skill8

Assuming that the expansion threshold remains at the fixed value of 0.8, there are the following GOZPDES parameters which can be changed that may influence the number of time steps and probability of knowing every skill that the students have at the end of the ZPD:

- The β and η values, determining the confidence in a new reward value for $w_a \leftarrow \beta w_a + \eta r$.
- The γ parameter, which determines the exploration rate when calculating the probability of selecting an activity to perform (Equation (2.21)).
- The ω and θ values determining when the ZPD can skip ahead and add a new activity when successfully answering hard activities.
- The d parameter, which determines the number of most recent answers used to calculate the reward value from an activity, as well as the minimum amount of times that an activity needs to be proposed before the ZPD can expand from it.
- The removal threshold - the success rate that an activity needs to have before it can leave the ZPD.

Table 4.5: Average probability of knowing every skill and number of time steps after going through the sequence of activities for students 1, 2, 3 and 4 having implemented all of the GOZPDES features.

Student1 time	Student2 time	Student3 time	Student4 time
402.5	115.3	134.5	290.0
Student1 % skills	Student2 % skills	Student3 % skills	Student4 % skills
97.8	99.7	99.5	98.2

4.4.4.A Results with default parameters

The first test performed with this new model, to be used as a baseline, contained the following parameter values:

- $\beta = 0.6, \eta = 0.4$
- $\gamma = 0.5$
- $\theta = 0.85, \omega = 0.8$
- $d = 4$
- removal threshold = 0.8

Table 4.5 shows the average number of time steps and probability of knowing every skill for the above GOZPDES parameters. There is a very large difference in the number of time steps between the students with higher and lower $baselearn$ values, this is due to the fact that the GOZPDES algorithm wants to ensure that the students know every skill by the end of the ZPD, even if it means having the worse students take a very long time to go trough it. On the other hand, every student has a probability of knowing every skill by the end of the ZPD of at least 97%. Given such a high probability, adjustment to the parameters should seek to decrease the number of required time steps, without compromising the probability of knowing all skills too much, and prioritizing decreasing the number of steps for the worse students. The values are rounded to one significant figure for both cases.

Henceforth, all parameters will remain with the values mentioned above, with the exception of the ones being tested at the time. All tests will be made with 20000 of each student type, for a total of 80000 students, as the tests will once again be made for students 1, 2, 3 and 4.

4.4.5 Optimal parameters identification

4.4.5.A β and η

The first parameters to be varied will be the β and η values, which determine the weight that a new reward will have. Since $\beta + \eta = 1$, then $\eta = 1 - \beta$ as the changes in the value of β are shown in Tables 4.6 and 4.7.

Table 4.6: Average number of steps to finish going through the sequence of activities for students 1, 2, 3 and 4 while changing the value of β.

β	Student1 time	Student2 time	Student3 time	Student4 time
0.0	404.3	116.2	136.1	293.9
0.1	405.6	116.0	135.7	291.4
0.2	405.3	115.5	135.3	292.6
0.3	404.0	115.6	134.8	292.5
0.4	404.8	115.5	135.3	291.2
0.5	403.9	115.1	135.2	290.8
0.6	404.8	115.3	134.3	292.6
0.7	403.9	115.3	134.3	289.1
0.8	399.0	115.5	133.8	289.9
0.9	396.5	115.6	134.2	285.9
1.0	395.5	116.6	135.8	287.3

Table 4.7: Average probability of knowing every skill for students 1, 2, 3 and 4 while changing the value of β.

β	Student1 % skills	Student2 % skills	Student3 % skills	Student4 % skills
0.0	97.7	99.6	99.5	98.3
0.1	97.6	99.6	99.4	98.1
0.2	97.8	99.7	99.5	98.2
0.3	97.7	99.7	99.5	98.1
0.4	97.6	99.6	99.5	98.2
0.5	97.6	99.7	99.4	98.3
0.6	97.7	99.6	99.6	98.3
0.7	97.9	99.7	99.5	98.2
0.8	97.7	99.7	99.6	98.1
0.9	97.5	99.7	99.5	98.2
1.0	97.7	99.7	99.6	98.1

Analyzing Tables 4.6 and 4.7 it seems that the ideal value of β is not the same for all students; for the students with lower $baselearn$, lower values of β seem to decrease the average number of steps without compromising the probability of knowing all skills, while the opposite happens for students with higher $baselearn$. This most likely happens because for the worst students it is important for the weight of an activity to increase quickly once they learn all of the required skills because it allows for these activities to be performed more often, removing them from the ZPD faster. This is not as important for students with high $baselearn$ values because of how quickly they learn skills, and is more sensitive to guesses in this case. If these divergences continue to occur for the other parameters, then the ideal GOZPDES parameters are different for different types of students, meaning that changing these parameters would be an excellent way of optimizing the learning experience for different types of students. Exactly how this should happen will be discussed further into this document. The β values that optimize the number of steps and probability of knowing all skills for the different students are as follows:

- Student1: 1.0
- Student2: 0.5
- Student3: 0.6
- Student4: 0.9

4.4.5.B Removal threshold

The next parameter tested was the removal threshold. However, the values obtained for the average number of steps and the probability of knowing every skill by the end of the ZPD when increasing the removal threshold to a value greater than 0.8 served no benefit to any of the students because all students would greatly increase the average number of time steps without gaining much in terms of the probability of knowing all skills. For students 2 and 3 this probability is already a minimum of 99%, so there is not much to gain, and for students 1 and 4 the exponential increase in the number of time steps does not justify the increase in the probability of knowing every skill, which is already at a minimum of 97%. However, thanks to activities not being removable from the ZPD unless all activities that have them as prerequisite have already been in it, it is possible to run tests for a removal threshold lesser than the expansion threshold of 0.8. The tests were made for removal threshold values between 0.5 and 0.75, as values lower than that offered too low a probability of knowing every skill for all students to be relevant. The values obtained for the removal threshold of 0.8 are available in Table 4.5, the results are in Tables 4.8 and 4.9.

Analysis of Tables 4.8 and 4.9 makes one thing quite clear: reducing the removal threshold can greatly decrease the average number of time steps for every type of student. However, this comes at

Table 4.8: Average number of steps to finish going through the sequence of activities for students 1, 2, 3 and 4 while changing the value of the removal threshold.

Removal threshold	Student1 time	Student2 time	Student3 time	Student4 time
0.5	286.2	86.3	99.7	207.5
0.55	295.6	88.9	102.3	214.8
0.6	301.1	90.3	104.6	218.9
0.65	313.0	93.0	108.0	227.9
0.7	330.9	96.7	112.2	239.6
0.75	356.6	102.0	119.3	257.2

Table 4.9: Average probability of knowing every skill for students 1, 2, 3 and 4 while changing the value of the removal threshold.

Removal threshold	Student1 % skills	Student2 % skills	Student3 % skills	Student4 % skills
0.5	64.9	94.0	90.4	70.9
0.55	84.1	97.9	96.6	87.8
0.6	85.6	97.9	96.7	88.3
0.65	90.6	97.6	97.7	92.3
0.7	92.1	99.6	97.9	93.3
0.75	92.2	99.8	98.1	93.6

the cost of decreasing the average probability of knowing all skills, raising the question of what is the minimum value of this probability that we are willing to accept in order to make this model faster to execute both in program and in the real world when user tests are made. For example, in order to keep a minimum average probability of knowing all skills of 92%, the ideal removal threshold for the different students would be as follows:

- Student1: 0.7
- Student2: 0.5
- Student3: 0.55
- Student4: 0.65

It may be worth to try to increase the probability of knowing all skills by changing other parameters, while keeping a lower removal threshold value in order to also maintain a low average amount of steps.

4.4.5.C d

The next parameter tested was d, which determines how many of the previous responses to an activity are used to calculate the reward, as well as the minimum number of tries required for a skill to be removable from the ZPD. Prior to performing the tests, the expected results was for small values of d to not be able to resist lucky guesses, leading to a lower probability of knowing all skills. Simultaneously,

Table 4.10: Average number of steps to finish going through the sequence of activities for students 1, 2, 3 and 4 while changing the value of d.

d	Student1 time	Student2 time	Student3 time	Student4 time
2	401.8	114.3	132.8	290.4
4	403.1	115.4	134.3	290.9
6	402.6	118.2	136.9	292.5
8	403.5	122.6	140.3	292.5
10	404.1	126.6	143.8	294.5

Table 4.11: Average probability of knowing every skill for students 1, 2, 3 and 4 while changing the value of d.

d	Student1 % skills	Student2 % skills	Student3 % skills	Student4 % skills
2	88.8	95.7	94.2	89.9
4	97.7	99.7	99.6	98.2
6	99.3	99.2	99.6	99.6
8	99.9	97.2	98.3	99.9
10	99.9	92.6	95.6	99.8

the average number of time steps and the probability of knowing all skills should increase as d increases. Tables 4.10 and 4.11 show the obtained results.

The results from Tables 4.10 and 4.11 are, in a way, somewhat unexpected. It seems that for larger values of d, the probability of the student knowing all skills actually goes down, and the better students (Student2 and Student3) have this happen to them for lower values of d than the worse students (Student1 and Student4). This likely happens because d also determines the minimum amount of times an activity has to be performed before the ZPD may expand downwards from it. Since the better students learn faster this means that a higher value of d is actually detrimental to them because it increases the likelihood that they will commit a slip after learning before the ZPD can expand. On the other hand, students that learn more slowly are more likely to have the ZPD expand due to a high success rate caused by guesses, so forcing a higher minimum number of attempts per activity helps to counteract this. Analyzing the numbers themselves, it seems that increasing d does not have a large impact on the average number of time steps, except for when the probability of knowing all skills is already decreasing. As such, the optimal value of d would be the one which provides the highest probability of knowing all skills, using the average number of steps only as a tiebreaker. Under this criterion, the optimal value of d for each student is as follows:

- Student1: 8
- Student2: 4
- Student3: 4
- Student4: 8

Table 4.12: Average number of steps to finish going through the sequence of activities for students 1, 2, 3 and 4 while changing the value of γ.

γ	Student1 time	Student2 time	Student3 time	Student4 time
0.0	415.6	114.3	142.2	305.9
0.1	416.3	122.9	139.9	300.7
0.2	412.0	119.7	136.7	297.6
0.3	408.7	117.4	136.1	294.8
0.4	404.9	115.3	134.8	292.6
0.5	404.1	115.1	135.0	292.0
0.6	402.6	115.3	134.3	290.8
0.7	399.5	115.8	134.3	286.3
0.8	396.2	115.5	135.0	287.8
0.9	395.0	116.7	135.4	287.1
1.0	396.0	117.0	136.4	289.3

Table 4.13: Average probability of knowing every skill for students 1, 2, 3 and 4 while changing the value of γ.

γ	Student1 % skills	Student2 % skills	Student3 % skills	Student4 % skills
0.0	98.3	99.7	99.6	98.5
0.1	98.2	99.7	99.6	98.5
0.2	98.1	99.8	99.6	98.4
0.3	97.7	99.7	99.5	98.3
0.4	97.7	99.6	99.6	98.2
0.5	97.8	99.7	99.5	98.2
0.6	97.8	99.7	99.5	98.1
0.7	97.8	99.6	99.4	98.2
0.8	97.5	99.7	99.5	98.1
0.9	97.5	99.6	99.4	97.9
1.0	97.8	99.7	99.5	98.1

Seeing as all of these students are rather extreme cases of good and bad students, it is likely that students with more intermediate $baselearn$ values will end up with an optimal d value of 6. That will be examined in greater detail further into this document. It is also important to mention that higher values of d should be used if the other parameters cause a low probability of learning all skills, as this may greatly increases that probability without compromising the average number of time steps.

4.4.5.D γ

The next value to be tested is the γ parameter, used in the adapted Exp4 algorithm to determine the influence that a uniform distribution has in the probability of choosing each activity in comparison to the activities' weights (Equation 2.21). The results are shown in Tables 4.12 and 4.13.

Analyzing Tables 4.12 and 4.13, it appears that the probability of knowing all skills doesn't change more than 1% across all γ values. However, the average number of time steps can be made to decrease considerably. As such, the goal with this parameter is to minimize the average number of time steps, using the higher probability of knowing all skills as a tiebreaker when the values are very close. The

optimal values of γ for students 1 through 4 are as follows:

- Student1: 0.9
- Student2: 0.5
- Student3: 0.6
- Student4: 0.7

The optimal values of γ differ with the $baselearn$ value of the student in question; students which are able to learn faster benefit more from intermediate values of γ, while students that take longer to learn skills benefit more from higher γ values, and since these lead to the probability of selecting a particular activity being more heavily influenced by a uniform distribution than the activity's weight, it means that for these students a random selection of the possible activities is the best way to progress through the ZPD, taking the least number of time steps without compromising the probability of knowing all skills.

4.4.5.E θ and ω

The last parameters to test are the θ and ω values which are used for skipping activities. In the original model this happened while keeping the activity that was already in the ZPD in it, but reducing its weight to 0. In this version the skipped activity is removed from the ZPD after the next time that it is performed. As mentioned before, this skip slightly decreases the average number of time steps for every student without decreasing their probabilities of knowing all skills. These two parameters were to be tested individually since, unlike with β and η, they have no direct relationship that allows to calculate one's value by knowing the value of the other. As a reminder, an activity can be skipped if a more difficult activity has a success rate of at least ω and this success rate multiplied by θ (with $\theta < 1$) is greater than the success rate of the activity to be skipped. However, after extensive testing with both values, it turns out that disabling the conditions related to θ and ω actually provided a greater improvement than the optimal values for these two parameters that were obtained through testing. A comparison is shown in Table 4.14, as it would seem that the prevention measures that are already taken to prevent improper advancement through the ZPD such as the minimum number of attempts caused by the d parameter and the precondition relation between certain activities are enough to prevent progress through guessing. Therefore, the activity skips can be done at a faster rate to improve the average number of steps without compromising the probability of knowing all skills, and the absence of the ω and θ parameters means that this particular part of the model works in the same fashion for every type of student.

Table 4.14: Average probability of knowing every skill and average number of time steps for Student1 with optimal θ and ω values versus after removing the conditions related to these parameters.

θ and ω Optimal time	θ and ω Optimal % skills	Removed time	Removed % skills
374.5	98.1	359.1	98.7

5

Evaluation

Contents

5.1 Simulator

This section contains a series of simulations performed in order to understand the impact that this work's students and exercises have on the accuracy of knowledge tracing.

5.1.1 Student comparison

One of the initial issues with the BKT-based initial version of the implementation was in the way of comparing the students with each other in order to discern the similarities between students. The first attempt was based on $P(L)$, also known as $P(KT)$, representing the estimated probability that a student knows a given skill. Initial tests calculated the n-dimensional Euclidian distance between each students, where n is the number of skills tested. The system had a greater than expected error rate (not choosing the most similar student).

As a reminder, students' learning speed can be related to their $baselearn$ value (Equation (4.1)) and their probability of knowing a skill before having a chance to learn it is influenced by the $baseinit$ attribute (Equation (4.2)). Figure 5.1 shows the effect that changing the value of x for Equation (4.1) when comparing 3 different students and attempts to understand what $baselearn$ value would be best in order to facilitate the distinction of students. The students used for this test were Student1, 2 and 3.

The comparison was made by having each student perform 8 exercises for 3 skills, with each exercise only being influenced by 1 skill. The Euclidian distance between the students for each BKT estimated probability of knowing the skill was then measured after the eighth exercise for each skill. The used skills were Skill1, 2 and 3.

The main parameter which determines the difficulty of a skill is $P(T)$, with the other parameters being more important to create different types of skills - a skill with a high $P(S)$ but low $P(T)$ could for example be a simple equation solving exercise, where the equation is easy to solve, but quite large, meaning that it is not difficult for students to accidentally make a mistake. Skills with very low $P(L_0)$ and high $P(T)$ are simple learning material related to somewhat obsolete knowledge which is often used as a learning step before moving into harder skills. The concept of skill is very much connected to this work's concept of exercise. However, skills are generally independent between themselves. The closest specification of skills which are harder than others is not necessarily related to each skill's parameters, as will be explained later.

With Figure 5.1 showing the probability of the system claiming that Student3 is more similar to Student2 than to Student1 after 200.000 executions (unless specified otherwise, every test from now on will have 200.000 executions), with $z = 0.1$. While it is clear that the students' individual parameters should influence their degree of similarity with each other, it seems that as x increases, so does the probability of the more similar student being correctly discovered. However, it is important to remember

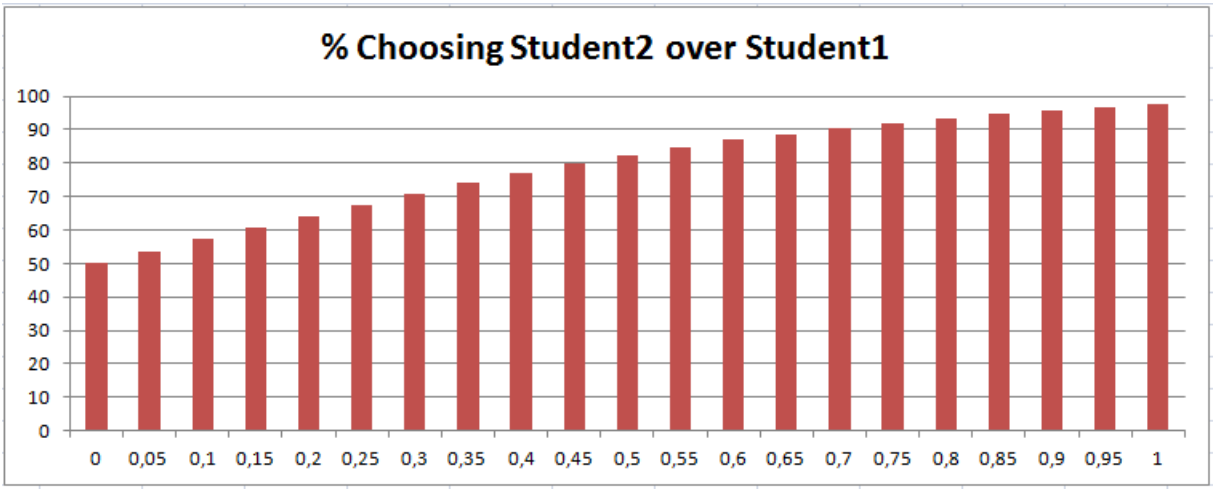

Figure 5.1: The probability that Student3 will be considered more similar to Student2 than to Student1, while changing the value of x from Equation 4.1.

that sometimes it is correct for the system to falsely consider less similar students as the closest two, as sometimes the expected most identical student will fail the exercises. Analyzing the results shown in the graph, the probability of choosing Student2 over Student1 steadily increases by 3 to 4 % until $x = 0.55$, at which point the growth in probability declines to about 2%, further decreasing as x increases. This indicates that $x = 0.55$ should be a solid value for the variable. It is important not to let the value of x be too low, as that would decrease the influence of the students' parameters, while at the same time raising x too high would disregard the difficulty of each skill. This means that the weight that the exercise difficulty has in student learning is in a way inversely proportional to the accuracy of the algorithm at selecting the most similar students.

Further analyses were performed regarding the accuracy of the student comparison using the Euclidian distance for $P(L)$ as criteria. The next test consisted of changing the number of skills, for $x = 0.55$ and $z = 0.1$. 8 exercises were performed for each skill, meaning that as the number of skills increases, so does the total number of exercises done by the students. The 3 students used were the same as for the graph in Figure 5.1. Skills number 1, 2 and 3 remained the same, adding in Skills 4 through 7.

For general use each exercise may use multiple skills, but for this test all exercises only handle one skill each. Results are shown in Figure 5.2 - it initially seems that increasing the number of skills and exercises is a good way of increasing the algorithm's accuracy. However, the probability of choosing Student2 over Student1 actually goes down on 6 skills. This may have happened because of the high $P(T)$ value for Skill6. The way to test the influence of the $P(T)$ value of each skill is to have the same students perform similar initial skills, followed by skills of varying difficulty. The graph in Figure 5.3 checks once again if Student2 or Student1 are given as more similar to Student3, except this time Skills 1 and 2 will be performed in that order, followed by a skill with varying value for $P(T)$ for each test, keeping

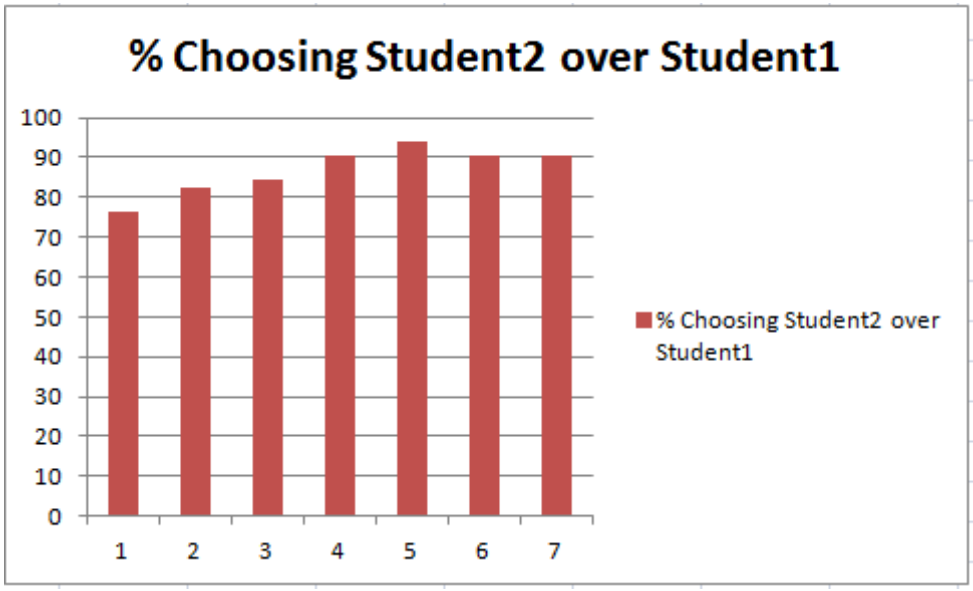

Figure 5.2: The probability that Student3 will be considered more similar to Student2 than to Student1, while changing the number of skills.

$P(L_0) = P(G) = P(S) = 0.05$. The results show that as $P(T)$ increases, the probability of Student2 being picked as more similar to Student3 than Student1 decreases slightly, ranging from 90 to 82 %. This is not a large difference, and it is likely not a good idea to give priority to higher difficulty skills when preparing new students before they are placed into a cluster, as this would likely make it more difficult to distinguish between different degrees of students with lower learning attributes. It is also unrealistic to presume that there are no easy to learn skills.

5.1.2 Multiple skills

When adding multiple skills to an exercise two new problems arise:

- How to calculate the probability that the student has of getting the exercise correct. This was simple for one skill as it was only needed to get the student's knowledge state for that skill and return $P(G)$ or $1 - P(S)$ depending on the situation.
- Exercises are much harder to get right the more skills are used in them, this may call for a new way of estimating students' knowledge.

Towards the first problem, several different methods of calculating the probability were used including the weighted average for each skill, only getting the exercise correct if the students solves a fake exercise for each of its skills, and simply using the least likely skill for the student to know as a criteria. The method that presented the highest accuracy in terms of selecting the most similar student at the end was the geometric mean (the square root of the product of the probability that the student was estimated to have

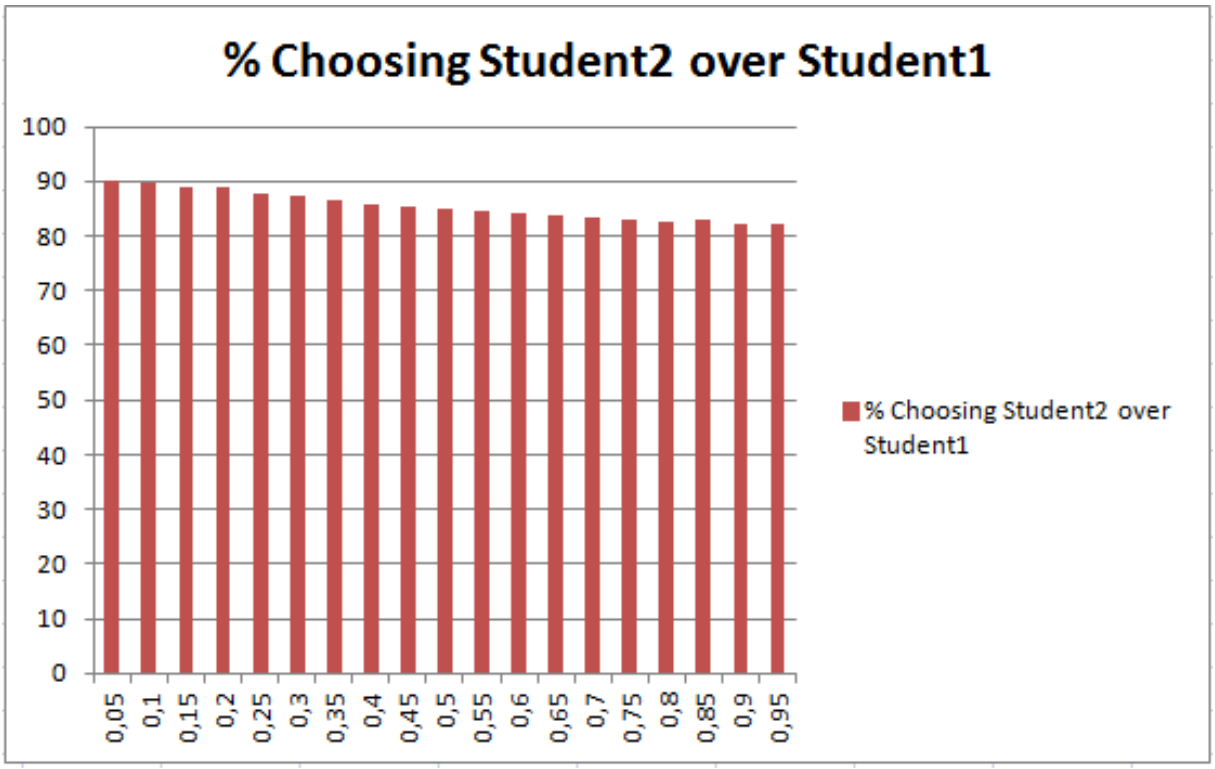

Figure 5.3: The probability that Student3 will be considered more similar to Student2 than to Student1, while changing the $P(T)$ value of the third skill being performed.

of knowing each skill). This does not necessarily mean that it is the best method of student comparison - it is expected that the two more similar students are not always correctly identified, as it is possible for one of the students to have abnormal results. This is essentially guaranteed to happen if the model is executed many times. Out of these methods, the two that show high enough accuracy to be considered are the geometric mean and the weighted average, with the weighted average being seemingly better due to increasing the percentage of selection of the correct students in comparison to when the criteria used is the BKT knowledge estimations, while keeping similar accuracy for the other methods.

5.1.2.A Impact of the number of exercises

For the second problem, further testing was done to compare the Euclidian distance based on the BKT estimate of skill knowledge for each student and two other methods: a simpler method - the percentage of correct exercises, as well as a slightly more complex alternative - the Euclidian distance based on the percentage of correct skill exercises, where a skill exercise for skill S is considered correct if the student correctly answers an exercise where the weight of S is greater than 0. The objective of the comparison was to see if any method provides consistently better results in terms of how often the right student is chosen. Figure 5.4 shows the probability of choosing Student2 over Student1 for both meth-

ods while changing the number of exercises, this should show how the number and variety of exercises affects the accuracy of these methods. Skills 1 through 7 remains the same, and Exercises 1 through 5 are added in numerical order.

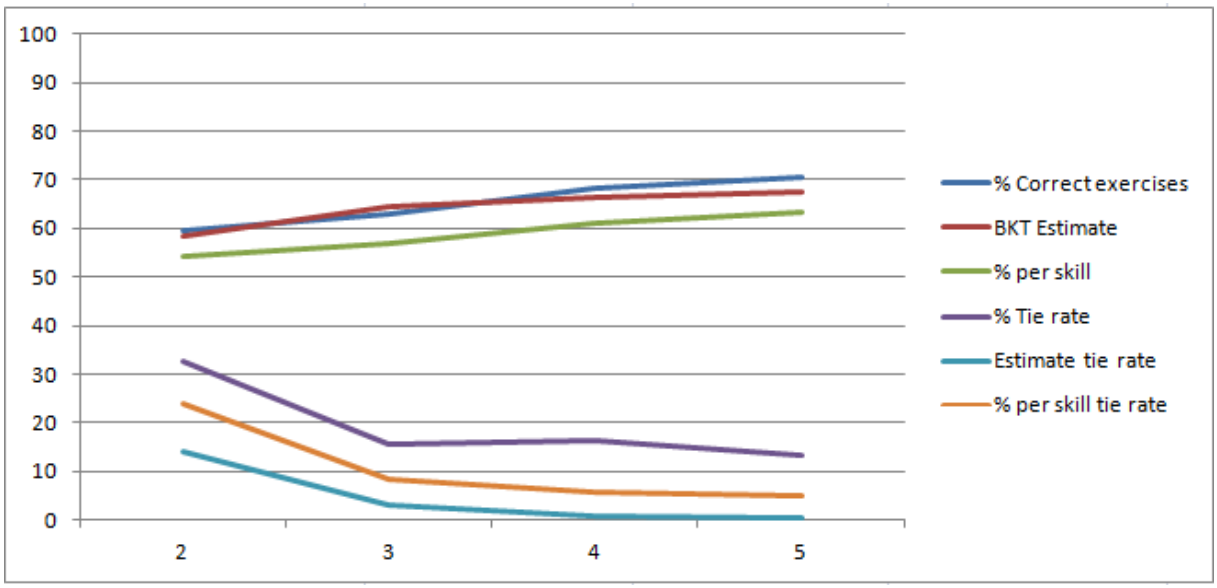

Figure 5.4: The probability that Student3 will be considered more similar to Student2 than to Student1, while changing the number of exercises done and changing the comparison method. .

Given the nature of Exercise1, the values only start being shown after both Exercises 1 and 2 are performed. Each student tries to solve each exercise twice, regardless of whether they answer right or wrong the first time, before moving on to the next exercise. This is done in order to give the students more learning opportunities. Students in this system do not treat solving the same exercise twice any differently from solving two different exercises with the same skills and weights. The tie rates (the probability that the method has of having at least two students which have the highest and equal degree of similarity to the student that they are being compared to) are also shown.

It is clear that the method using the Euclidian distance based on the percentage of correct answers per skill is strictly worse than the one measuring the Euclidian distance based on the BKT probability estimations, as it consistently has a lower chance of picking the two most similar students, and it also has consistently more ties. This is most likely due to the fact that the weight that each skill has in an exercise is not being taken into account by this method. This leaves the comparison between the remaining two methods in Figure 5.4. The percentage of correct exercises is a much simpler method that seems to be about as accurate a method as the BKT estimate. However, it has a higher tie rate, 33% to 15%. This is due to the fact that this method completely ignores what skills are used in each exercise and cares only about whether the answers are correct or not, as well as this method's simplicity. The BKT based method has the disadvantage that after the student gets two answers correct for a certain skill there

is very little difference in its estimated probability of whether or not the student knows the skill when he or she get more correct answers, while the percentage of right answers method makes a clearer distinction between students who learn a skill quickly and those who take several exercises to learn it. This is good at distinguishing the good students from the bad ones, but it may not be so accurate at distinguishing worse students, or when several similar students exist. Therefore, a test to measure how accurately these methods are when comparing students who had more difficulty at learning the skills shall be made. Before that, however, different tests will be made to see if these methods are able to remain accurate if the number of students to compare increases, as well as to attempt to find a way to break ties in a favorable manner.

In this system, a tie happens when the method used to compare students considers that there are at least two equally similar students to the one being used as a base for comparison. According to Figure 5.4, the percentage of correct exercises method presents a relatively high tie rate - a minimum of 13% - if some of these ties could be converted into correct predictions, then the accuracy of the algorithm would increase. Tie breaking is a difficult procedure, as in this case there are several hidden variables at play, and whatever tie breaking method is used must deal with distinguishing between students with very similar results.

An initial attempt to deal with some of these ties was to have the tied students and the base student perform an additional fake exercise, and if they don't all answer correctly or incorrectly, state that the most similar student to the base student is the one who answered the same way. This technique, however, did not increase the accuracy of the algorithm, so it was discarded. Another attempt was to use the number of skills that the students knew as a tie breaker. This presented negligible improvements in the accuracy and tie rate, and wasn't very realistic, as it is assumed that the students are always right in what skills they claim to know. An attempt at having the students very rarely making a mistake in regards to claiming whether or not they know a certain skill actually led to a decrease in accuracy, so this method was also discarded.

For the next set of tests, a new student is added: Student4. The first test will be a repeat of the previous one, having added Student4 to the group of students. This student has twice the $baselearn$ value of Student1, but is still not a very good student. The goal of this test is to see if the comparison methods being used are able to still accurately compare the students. The Euclidian distance based on the percentage of correct exercises per skill was also included for the sake of comparison. The random guess threshold in this case is $\frac{1}{3}$. The results are shown in Figure 5.5.

Comparing the results in Figure 5.5 with those in Figure 5.4, it seems that the Euclidian distance based on the percentage of correct answers per skills method is actually a competitive method with this additional student, despite still having a lower accuracy and higher tie rate than the BKT-based method after 3 or more different exercises are made. As for the other two, both of them have lost

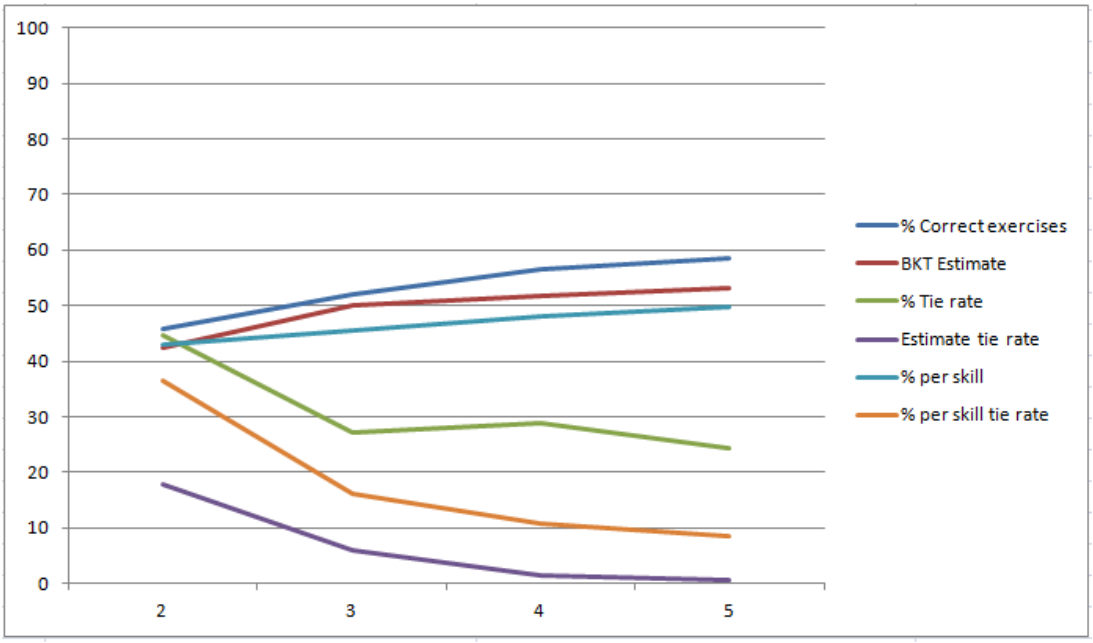

Figure 5.5: The probability that Student3 will be considered more similar to Student2 than to Student1 or Student4, while changing the number of exercises done after Exercise 1 and changing the comparison method.

accuracy compared to Figure 5.4, while the percentage of correct exercises method ends up having an approximately 10% higher tie rate for all numbers of exercises.

5.1.2.B Comparison for slow learners

The next test performed returns to having 4 students, but this time the comparison is made having Student4 as a base and seeing how often Student1 is chosen as most similar, running the same 2 to 5 exercises as before twice. This test should give a better idea of which methods are better at making distinctions between bad students and students with some difference in quality, as Student4 has twice the $baselearn$ value of Student1, and just over a third of the $baselearn$ value of Student3.

The results shown in Figure 5.6 are somewhat unexpected - it appears that the Euclidian distance methods are not good at dealing with comparing students with lower baselearn values. In fact, further increasing the number of exercises does not increase the accuracy of these methods. However, looking at the percentage of correct exercises method, its accuracy was practically the same as it was in Figure 5.5. This leads to the conclusion that this method is most likely the best of the three when it comes to comparing several different students, as the only time when it shows a considerable decrease in accuracy is when the number of students increases, a problem which is manageable by increasing the number of exercises done, and it is possible that in the future its higher tie rate can be handled in a way that further increases accuracy. Henceforth, the Euclidian distance based on the percentage of correct exercises per skill method will no longer be shown in future graphs, as it consistently has the worst

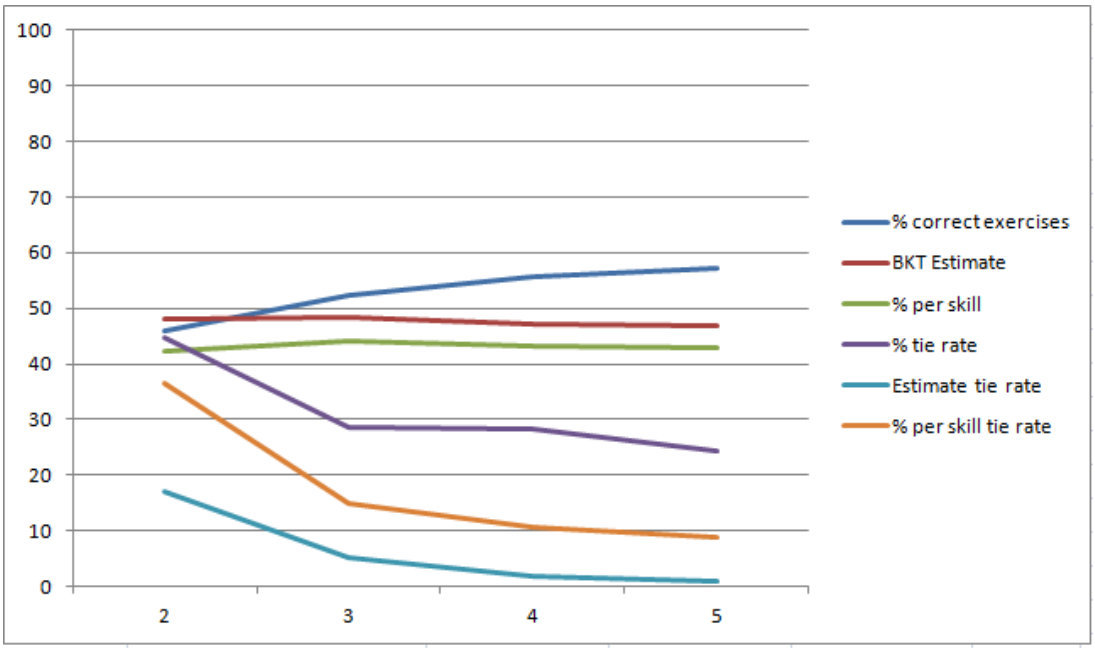

Figure 5.6: The probability of the skill estimate and percentage of correct exercise methods of stating that Student1 will be considered more similar to Student4 than to other students, changing the number of exercises.

accuracy out of the three methods, while the Euclidian distance based on the BKT estimate method will still be shown for comparison.

5.1.2.C Impact of changing the number of students

This calls for a new test to see exactly how these methods scale with the number of students: this time always performing the same 5 exercises, but with a varying number of students - up to 10. Students 1 through 10 are now used.

All of the added students are in theory less similar to Student3 than Student2 is, but with the increase in the number of students to compare it is more likely for errors in classification to occur in the comparison algorithm.

Upon examination of the results in Figure 5.7, it is clear that the number of students being compared has a negative effect on how accurate every method is, as well as in their tie rate with the exception of the BKT based method of comparison. Out of these three methods, the one that suffers the least from the increasing amount of students in terms of accurately selecting the right student is the percentage of correct answers method.

On a side note, increasing the amount of times that an exercise is repeated, increasing the total amount of exercises done, has an overall positive effect on the accuracy of every method, so the number of exercises done should be increased along with the number of students in order to maintain the accuracy of these algorithms.

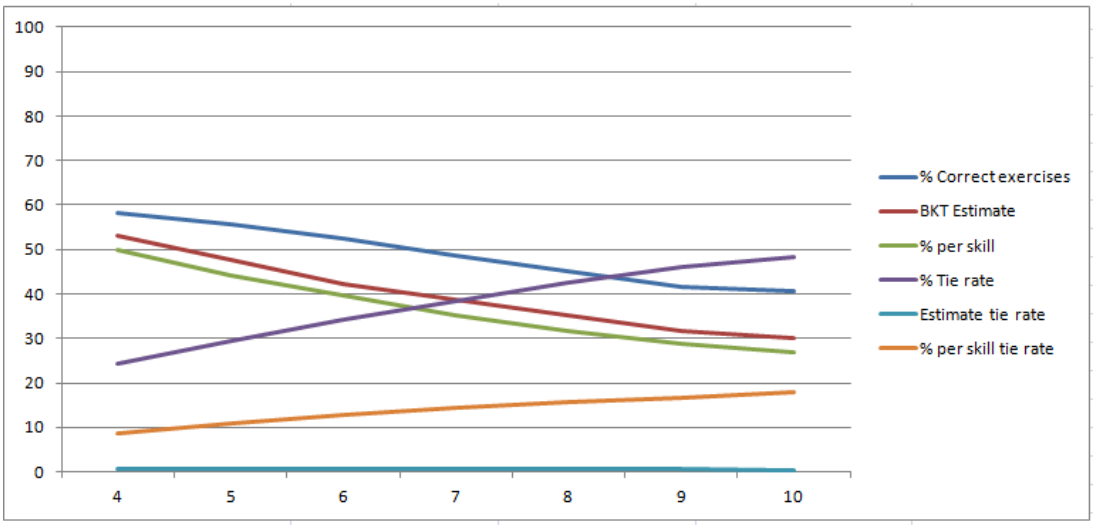

Figure 5.7: The probability of the skill estimate and percentage of correct exercise methods of stating that Student3 will be considered more similar to Student2 than to other students, changing the number of students.

5.1.2.D Student distribution analysis

At this point the comparison of students has been thoroughly studied, so the focus will temporarily change into the ability that each skill has of assisting in the distinction between different kinds of students - a sequence of exercises that use different skills with different weights does not necessarily imply that every skill is being helpful in distinguishing between students. In order to verify if that was the case a smaller scale experiment was made - Students 1 through 4 performed Exercises 1 through 5 twice for 20.000 iterations, and the final estimated probability of each student knowing each skill using the BKT-based method was exported to Attribute Relation File Format (ARFF), a file format customized for use by Waikato Environment for Knowledge Analysis (WEKA) [47]. This brings the additional advantage that it provides access to all of WEKA's clustering and classification methods including EM, K-means and KNN.

Figures 5.8 and 5.9 show the distribution of students according to their estimated probabilities of knowing Skill3. Figure 5.8 for when only Exercises 1 through 5 are done twice, and Figure 5.9 for when the two following exercises are also added:

- Exercise6: only uses Skill2, weight 0.8
- Exercise7: uses Skills 3 and 4
  - Skill3 weight: 0.7
  - Skill4 weight: 0.3

Each color in Figures 5.8 and 5.9 correspond to a different type of student: light blue corresponds to Student1, Student2 is red, Student3 is dark blue, and Student4 is gray. Their distribution across

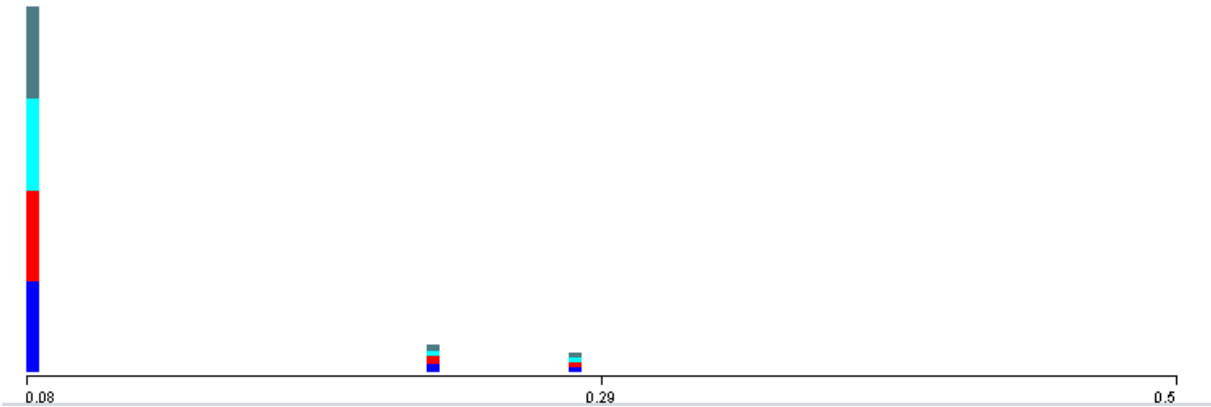

Figure 5.8: Distribution of students by estimated probability of knowing Skill3

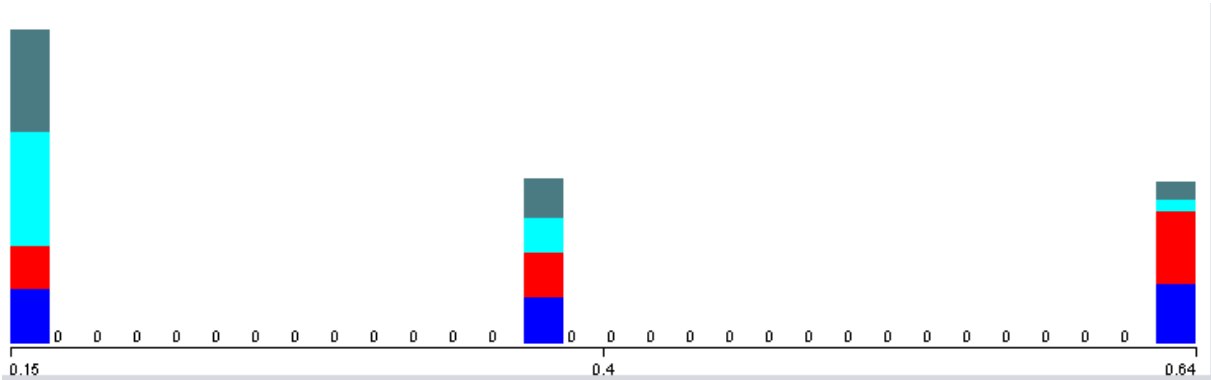

Figure 5.9: Distribution of students by estimated probability of knowing Skill3 after Exercises 6 and 7 are added

different estimated probabilities of knowing the skill is much more uniform before Exercises 6 and 7 are added, and the same was happening on Skill2. It is clear that adding these exercises has changed the student distribution, with the two students with the lower $baselearn$ values being mostly in low values of estimated probability of knowing the skill. Students 2 and 3, who have much higher $baselearn$ values, are the majority of the students who reach high values of this estimated probability. Having added these two additional exercises, their distribution becomes very similar to the one shown in Figure 5.9 for most skills, meaning that the other previously existing exercises already did a good job of distinguishing good from bad students. Adding students with intermediate $baselearn$ values also had the expected outcome: there are less of them in the lower estimated probabilities of knowing a skill than the worse students, but they are also less present in the higher estimated probabilities than the students with higher $baselearn$. At this point more exercises were added to ensure that the distribution of students across the different skills is able to distinguish between different types of students, and also spread them across the whole $[0, 1]$ range of the probability.

5.2 Student answer probabilities

Having found a good way to observe the distribution of students, the next task was to implement a variant of ZPDES that took advantage of this information to be able to distinguish between students, while still making sure that they are learning the skills. While the previous work with this algorithm uses the concept of KCs as what the students are supposed to learn, this work's concept of skills is similar enough to be used instead, especially since the most important part of this work is the exploration of the advantages that student grouping may offer, and not necessarily how accurate the representation of the learning components is. The main disadvantage of this choice is that different skills do not directly influence each other. However, this can be mitigated by forming ZPDES graphs with dependencies between activities which use few skills. It also allows for the use of weights that each skill has for each activity and allows the work to focus on the activities themselves.

It is important to mention that the goal that ZPDES was designed for is to teach a series of KCs to students as efficiently as possible, making sure that the students have learned every skill by the end of the algorithm's execution. This differs from the work that's been done so far in that the main goal was to differentiate between students. This means that attempting to use ZPDES for that task would come with several limitations. That being said, it is not impossible - for example, the number of time steps that it takes for a student to go through all of the proposed exercises is a metric that may be used to distinguish between good and bad students, while still ensuring that they all learn each skill. The way that it is evaluated whether or not the student correctly answers an activity is given by Equations (5.1) and (5.2), where $P(C)$ is the probability that student st will get the activity correct, sk is a skill of activity A (A is a set of skills for these equations), $P(G_{sk})$ is sk's guess parameter, $P(S_{sk})$ is sk's slip parameter and γ_A is an auxiliary parameter.

$$\gamma_A = \prod_{sk}^{A}(K(st) \times P(G_{sk}) + (1 - K(st)) \times (1 - P(S_{sk}))), K(st) = \begin{cases} 0, & \text{if st knows sk} \\ 1, & \text{otherwise} \end{cases} \tag{5.1}$$

$$P(C) = \exp(\log(\frac{\gamma_A}{|A|})) \tag{5.2}$$

After performing an activity, the student has an opportunity to learn every skill that was used, meaning his knowledge of the skill may transition from the unlearned to the learned state. The probability of student st learning a skill sk after performing an activity (or exercise) A which includes it is given by Equation (5.3):

$$P(T) = (0.55 \times baselearn_{st} + 0.45 \times P(T_{sk})) \times w_{skA} \tag{5.3}$$

Where w_{skA} is the weight that skill sk has in activity A, meaning that for GOZPDES the weight of a skill in an activity only affects how likely the student is to learn it after performing said activity, and therefore it only indirectly influences how likely the student is to get the exercise correct. The number 0.55 was obtained from previous analysis of the graph in Figure 5.1. The original ZPDES algorithm did not have the concepts of $baselearn$ and skill weight.

For the context of this work, an activity works similarly to an exercise, but its success rate is updated with each right or wrong answer, and each activity may be connected to other activities in an oriented tree structure, or polytree [48], with the arrows indicating which activities the ZPD may update into. Regardless of the structure of the activities, only those in the ZPD have a chance of being proposed to the student. Some activities may be prerequisites for other activities, but it is also possible for activities with no prerequisites to be connected, which is particularly important for updating the ZPD in cases where prerequisites between activities are not being met.

5.3 Profile students/Student profiles

A profile student is no more than a student that has a GOZPDES model with optimal parameters associated to it, meaning that the given model is optimal for all students with a similar $baselearn$ value, and since it is possible to estimate this parameter, it is also possible to provide students with a GOZPDES with parameters which are approximately optimized for their learning speed. The way that these parameters were determined was through an exhaustive series of manual parameter adjustment. Since students' success is measured using the average number of time steps required to make it through an execution of the algorithm and the average probability of knowing all skills by the end of it, the goal is to either minimize the required average time steps while keeping the probability above a certain threshold or to maximize this probability while keeping the average number of time steps below a certain value. In this case the chosen threshold was a minimum probability of learning all skills of 97% because it provides ample opportunity to reduce the average number of time steps while still making sure that the vast majority of students end up knowing all skills. Since the parameters are manually optimized, it would be possible to in the future select a different optimal GOZPDES based on different probability thresholds or maximum number of time steps, allowing for students or teachers to personalize the use of this algorithm to best suit their needs and constraints.

Student profiles were created for $baselearn$ values from 0.1 to 0.9 in increments of 0.1, for a total of 9 profiles. All ZPDES parameters with the exception of the expansion threshold were tested with different values in order to determine the optimal GOZPDES model for each profile. Ultimately, the optimal parameters for each student profile (the nomenclature used for them is P_x, where x is that profile student's $baselearn$ attribute) are shown in Table 5.1, along with the time taken by each profile if

Table 5.1: Optimal parameters and average number of time steps for a series of student profiles along with a comparison to using the optimal parameters for $P_{0.1}$ and $P_{0.4}$. Results with a * afterwards did not reach the 97% threshold.

Parameter	$P_{0.1}$	$P_{0.2}$	$P_{0.3}$	$P_{0.4}$	$P_{0.5}$	$P_{0.6}$	$P_{0.7}$	$P_{0.8}$	$P_{0.9}$
Rmv thrshld	0.6	0.6	0.6	0.55	0.55	0.55	0.55	0.55	0.55
β	0.9	0.9	0.9	0.8	0.7	0.65	0.6	0.5	0.45
d	8	8	6	6	6	6	4	4	4
γ	0.2	0.1	0.2	0.25	0.35	0.4	0.45	0.45	0.5
Optimal time	271.9	202.9	160.7	133.5	116.9	105.0	89.5	81.9	76.0
$P_{0.1}$ time	N/A	204.6	166.7	142.3	125.9	114.3*	105.4*	98.5*	92.9*
$P_{0.4}$ time	264.9*	195.5*	157.6*	N/A	117.3	105.7	96.0	88.8	83.4

using the optimal parameters of $P_{0.1}$ or $P_{0.4}$:

In addition to the student profiles taking longer when not using optimal parameters in Table 5.1, $P_{0.6}$, $P_{0.7}$, $P_{0.8}$ and $P_{0.9}$ did not reach the 97% threshold when using the optimal parameters for $P_{0.1}$. Analyzing this table's results in terms of the difference in the average number of time steps, it seems that the closer the profile's optimal parameters were to the optimal $P_{0.1}$ parameters the less difference there is in regards to time. This means that eventual small errors in the estimation of $baselearn$ and classification of students based on this parameter are not very harmful for the student's performance. Additionally, the time required for the students under the optimal parameters for $P_{0.4}$ shows that for Students $P_{0.3}$, $P_{0.5}$ and $P_{0.6}$ there is little difference between their optimal parameters and the "default" parameters. Given this similarity between optimal values for students with approximate $baselearn$, an eventual error in selecting the optimal parameters would likely be of little consequence, as the optimal parameters for an almost equivalent student profile are very similar.

This paradigm of optimizing the model through the creation of predetermined student-like objects, followed by assigning students to their most similar optimal object is in a way equivalent to the philosophy taken by machine teaching, with the focus on minimizing the size of the data set being replaced by the minimization of the average number of time steps or the maximization of the average probability of knowing all skills, depending on what criterion is selected as the threshold.

Regarding the progression of the optimal parameters, the main influence in terms of number of time steps and probability of knowing all skills are the removal threshold and d parameters, with the β and γ values being simply changed in order to be the most optimal for each student profile, which have parameter values for each student profile that is optimal in both average number of time steps and average probability of knowing all skills. The only parameter that has some unexpected behavior is γ, which is optimal at small values for students with lower $baselearn$, and it increases up to more intermediate values for the better students. This last part was expected, but the previously obtained optimal values were higher γ for students which take longer to learn. This may be because of the criteria that was previously used to determine the optimal value for each student, which focused on minimizing

the amount of time steps and ignored the probability of knowing all skills.

In case of ties when selecting the optimal parameters the chosen student profile is always the one recommended for students with lower $baselearn$. This is done to maximize the probability that every students has of learning the proposed skills, at the cost of slightly increasing the number of time steps. It would also have been possible to do the opposite for ties, which would lead to students taking slightly less time to go through the model at the cost of a small decrease in the average probability of knowing all skills.

In addition to the above test and in an attempt to understand if the estimations made are accurate enough for optimizations based on them to be preferable to a set of default parameters, a series of students with $baselearn$ attributes between 0.1 and 0.9 (in increments of 0.1) had their competence estimated and were given a GOZPDES model with optimal parameters for the most similar student profile. This was repeated 20 times. All of them selected the appropriate optimal parameters. Therefore, Table 5.1's values are still valid.

5.4 Student classification process

As it was previously shown, students with different $baselearn$ values have different optimal GOZPDES parameters. It should be possible to take advantage of this fact in order for each student to have a more personalized learning experience. However, the previously performed test had knowledge of the actual $baselearn$ value of each student. In a real world scenario, it is unreasonable to expect a student to be able to estimate its own competence. It is therefore necessary to create a way to estimate each student's $baselearn$ attribute. As a reminder, Equation 5.3 calculates the transition probability between the unlearned and learned state for student st and skill sk. This Equation can also be used to calculate a student's $baselearn$ if $P(T)$ is already known. Equation 5.4 shows Equation 5.3 solved for $baselearn$.

$$baselearn = \frac{P(T) - 0.45P(T_{sk})w_{skA}}{0.55w_{skA}} \tag{5.4}$$

Thanks to the Baum-Welch algorithm, it is possible to reestimate the value of $P(T)$. The used implementation of this algorithm was obtained from the jahmm library [46], which also contains a Java implementation of HMM.

The way to take advantage of this re-estimation capability in an environment that uses virtual students is the following:

1. Create two different HMMs: one that models the probability of transitioning from the skill's unlearned state to the learned state as $P(T_{sk})$ and one using $P(T)$ instead, which is obtained via Equation (5.3)

2. Generate a sequence of observations from the HMM that uses $P(T)$
3. Compute the Baum-Welch algorithm for the HMM that uses $P(T_{sk})$, but use the observations obtained in the previous step
4. Extract the estimated $P(T)$ value and use it to recalculate $baselearn$ through Equation (5.4)
5. Create a GOZPDES model using the optimal parameters for the profile students with the closest $baselearn$ value
6. Go through the GOZPDES algorithm normally

This procedure should also be able to estimate a real student's $baselearn$ value, with the exception that no HMM is specifically generated for the student and the observations must be obtained from the student performing exercises instead of generated, which would certainly lead to having a much smaller number of observations to work with. The value of $P(T_{sk})$ can be known in a real world scenario either from previous tests or from an expert's estimate.

The obtained estimated $baselearn$ can then be used to determine what group the student will belong to. For this, some type of grouping criteria must be defined beforehand, which is the purpose of the student profiles.

5.5 Impact of classification errors

Having determined the optimal parameters and possessing a way to estimate the students' learning speed through re-estimation of the $baselearn$, it is now time to see if it is possible to have the virtual students take advantage of these techniques. The next test shall create students with random $baselearn$ values, estimate the attribute, assign them the optimal GOZPDES parameters of the student profile most similar to them and finally run the GOZPDES algorithm for the students using these parameters. The results are shown in Table 5.2, along with the average number of steps for both each corresponding profile's optimal parameters and the previously established default parameters, along with the probability of each student profile knowing all skills when using default parameters. It is worth mentioning that the average number of steps and probability of knowing all skills for the random students is sorted by which profile they were assigned as most similar, meaning that errors in classification are possible. This is done deliberately as these errors can certainly occur for real students and the obtained results are worth comparing to those obtained when in the optimal scenario, where only profile students are used and the classification is perfect.

Along with the results of Table 5.2, the average parameters obtained were the following:

- $d : 5.628$

Table 5.2: Average number of time steps and probability of knowing all skills for students with random $baselearn$ values, sorted by which profile student was considered to be the most similar to each student, along with the average number of time steps for optimal and default parameters, as well as the probability of knowing all skills for each profile when using default parameters. The results show that the random students are on average less likely to learn all skills, but are faster at going through the ZPD

Parameter	$P_{0.1}$	$P_{0.2}$	$P_{0.3}$	$P_{0.4}$	$P_{0.5}$	$P_{0.6}$	$P_{0.7}$	$P_{0.8}$	$P_{0.9}$
Num steps	321.6	235.0	179.0	158.2	134.8	124.7	106.7	98.1	89.7
% all skills	99.3	97.5	92.5	97.3	90.6	86.6	98.4	93.8	91.6
Default steps	364.3	267.2	213.1	179.2	155.5	139.2	126.3	116.3	108.4
Default % skills	99.9	99.9	99.8	99.5	99.0	98.5	97.6	96.5	95.3
Optimal time	271.9	202.9	160.7	133.5	116.9	105.0	89.5	81.9	75.9

- Remove threshold: 0.566
- $\beta : 0.688$
- $\gamma : 0.339$

Since d is necessarily an even number, it is rounded up to 6.

Analysis of Table 5.2 shows that comparing to the default parameters, the estimated optimal parameters greatly decrease the average number of time steps at the expense of a penalty in the probability of knowing all skills. Still, with the exception of the students assigned as most similar to profile student $P_{0.6}$, the average probability for all students is a minimum of 90%. Overall, these results are better than the ones obtained when using the default parameters. For example, the random students assigned as most similar to profile student $P_{0.7}$ show better results in both of the observed values than the respective profile student using the default parameters. Comparing to the profile students using their optimal parameters, the number of time steps is, as expected, lower than the ones obtained by the random students.

So far, the estimation of the $baselearn$ attribute is obtained before the students actually perform any activity by generating observations through a diagnostic test of sorts. While this approach can also be used on real students, the number of time steps taken does not take this test into account. It would likely be advantageous to find a way to skip the test, allowing the estimation of the $baselearn$ attribute to be made dynamically while the student is performing the activities within the GOZPDES algorithm.

5.6 Online grouping

The next test shall measure the average number of time steps and probability of knowing all skills when initially using default parameters, which are the average parameters obtained from Table 5.2. Several different conditions determining when the Baum-Welch algorithm is executed are tried. There is no specific need to check whether or not the correct optimal parameters are estimated, as Table 5.1 also

Table 5.3: Average number of time steps and probability of knowing all skills, starting with default parameters and updating them dynamically at certain points through the model's execution with the activities shown in Figure 4.1. The conditions tested are performing the estimation after a certain number of time steps, after A1 has been removed from the ZPD, whenever A1 or B1 are removed from the ZPD and using the average parameters throughout

Condition/Profile	$P_{0.1}$	$P_{0.2}$	$P_{0.3}$	$P_{0.4}$	$P_{0.5}$	$P_{0.6}$	$P_{0.7}$	$P_{0.8}$	$P_{0.9}$
20 steps time	283.9	210.1	177.4	150.6	137.7	123.0	116.4	107.7	102.5
20 steps %skills	98.5	98.2	97.1	96.2	95.8	93.3	91.1	90.5	86.5
30 steps time	282.6	211.1	177.8	154.6	137.2	122.5	114.7	106.3	101.3
30 steps %skills	97.9	98.2	97.4	96.8	94.0	93.3	92.5	89.6	88.0
A1 time	285.0	216.2	182.4	157.0	141.2	125.7	116.9	106.3	101.0
A1 %skills	98.5	98.1	96.8	96.0	94.5	92.9	88.4	88.5	86.1
A1/B1 time	285.0	223.4	181.0	162.4	143.7	130.4	120.3	114.5	107.0
A1/B1 %skills	99.0	98.0	96.9	95.1	92.5	89.2	86.5	85.0	81.7
Avg params time	284.2	211.8	181.2	157.5	140.6	129.6	120.2	114.3	108.7
Avs params %skills	98.0	97.9	96.8	95.9	91.4	89.2	84.2	81.8	77.1
30 steps+ time	282.5	208.6	173.9	147.4	130.9	117.2	109.0	101.9	95.3
30 steps+%skills	97.5	98.5	97.4	97.7	95.9	96.8	94.1	91.5	90.9

shows that the average number of time steps and probability of knowing all skills are negatively affected when suboptimal parameters are used, so these are the only values that need to be shown in order to measure the quality of the different conditions for re-estimation.

The procedure used for each student in this case is the following:

1. Go through the GOZPDES algorithm normally using a set of parameters which is initially the same for every student and saving the answers' correctness. This continues until the estimation condition is met

2. Once the estimation condition is met, create an adequate HMM. The $P(T_{sk})$ value used depends on the condition being tested

3. Compute the Baum-Welch algorithm using the HMM generated in the previous step and the observation sequence obtained from the first step

4. Extract the estimated $P(T)$ values and use it to recalculate $baselearn$ through Equation (5.4)

5. Change the GOZPDES parameters to be the optimal ones for the profile student with the closest $baselearn$ value

6. Go through the updated GOZPDES algorithm normally

The results are shown in Table 5.3.

The results in Table 5.3 show that dynamically determining the optimal parameters for each student is a viable option. Comparing to the values obtained in Table 5.1, the average number of time steps only increases slightly, most likely due to the fact that there is always a period of time when suboptimal

parameters are being used, but this is not an issue since the diagnostic test no longer needs to be performed and the number of time steps that that test required were not being taken into account, so the actual number of time steps required has not increased much. The main difficulty that this strategy seems to have is the comparatively lower value of the probability of knowing all skills for above average students.

Regarding the conditions tested it seems that the simpler ones tend to show better results, with the estimation happening after both A1 and B1 have been removed from the ZPD being on average both slower and less effective at teaching than simply doing the estimation after A1 leaves the ZPD. Again, this most likely happens because the student ends up using suboptimal parameters for a higher number of time steps. This also shows the problem that a system that tries to treat all students differently would have - instead of the estimation of a single student attribute, the required approach would try to find all of the optimal GOZPDES parameters per student, which is a complex problem because these parameters are not necessarily independent of each other, meaning that an algorithm like Exp4 would have to be executed multiple times to determine each individual parameter value. This process would take a great number of time steps to execute properly, but if well implemented it would be able to update the parameters after each exercise. It is unclear whether or not this would offset the issues, but seeing how the results in Table 5.3 show that the speed of the estimation can be just as important as the quality of the chosen condition that seems rather unlikely since the MAB-based estimation would require either a very fast adaptation to new rewards, which may lead to actually worsening the parameters depending on how each activity is performed, or a slower adaptation which would likely take too many time steps to be efficient in this environment.

Out of the conditions shown, the one that penalizes these students the least is the parameter update after performing 30 time steps. Additionally, other not shown conditions were attempted such as attempting to update the parameters every 20 or 30 time steps, but these strategies did not show better results than the ones that only perform the update once, and also greatly increase the program's execution time due to the computational complexity of the Baum-Welch algorithm.

After all other strategies were tried, and having reached the conclusion that the main problem that the faster learning students were having was that their $baselearn$ attribute was being underestimated, a slight adjustment to the estimation algorithm was made, after which the estimation after 30 steps was attempted again. The results are available in Table 5.3 as the "30 steps+" condition, and it is noticeable that there were improvements in the average amount of time steps and probability of knowing all skills for every type of students except for the ones with the lowest estimated $baselearn$.

Figures 5.10 and 5.11 show the decrease in the average number of time steps and increase in the probability of knowing all skills respectively for using the "30+", after 20 time steps, removing activities A1 and B1 from the ZPD for the different student profile values or the a priori estimation in comparison

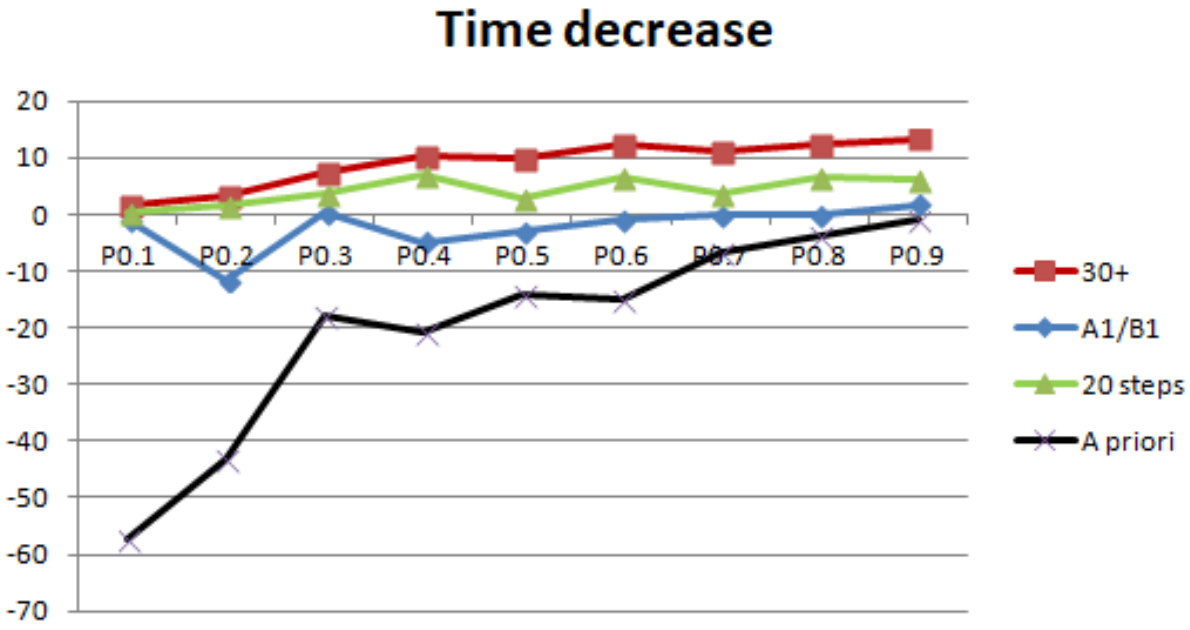

Figure 5.10: Time gain by different conditions and a priori estimation in comparison to using average parameters from the start

to simply using the average parameters throughout. The a priori method has a 20 time steps penalty which corresponds to the time taken to perform a diagnostic test. Figure 5.10 shows a decrease in the number of time steps for the "30+" and removing A1 and B1 methods, with the "30+" condition being the better method. While there is little to no gain for the worst students, as the average $baselearn$ value of the students increases, the average number of time steps decreases further. The same can be said for the improvements in the average probability of knowing all skills - there is little to no improvement for the worse students, but the better the students are the greater the improvement is. This most likely happens because the GOZPDES algorithm naturally causes the worse students to stay within it for such a long number of time steps that they are essentially guaranteed to learn every skill by the time that they leave the ZPD, allowing for essentially no gain in that regard. Simultaneously, there is little difference on the number of time steps for the slower students because the main responsible for their slower progress through the ZPD is the expansion threshold, which is deliberately the same for all students in order to ensure that slow learners are unable to progress via lucky guesses.

In comparison to the a priori $baselearn$ estimate every other method takes a lower number of time steps due to not suffering the penalty for performing the diagnostic test. In terms of gain in the probability of knowing all skills, however, the "30+" condition is able to have considerably better results for average students while being almost as good for the faster learning students. The superiority of the a priori method is mostly shown there, as the optimal parameters for the best students are quite different from the average parameters which are used until the reestimation condition is met (Table 5.1). This necessarily

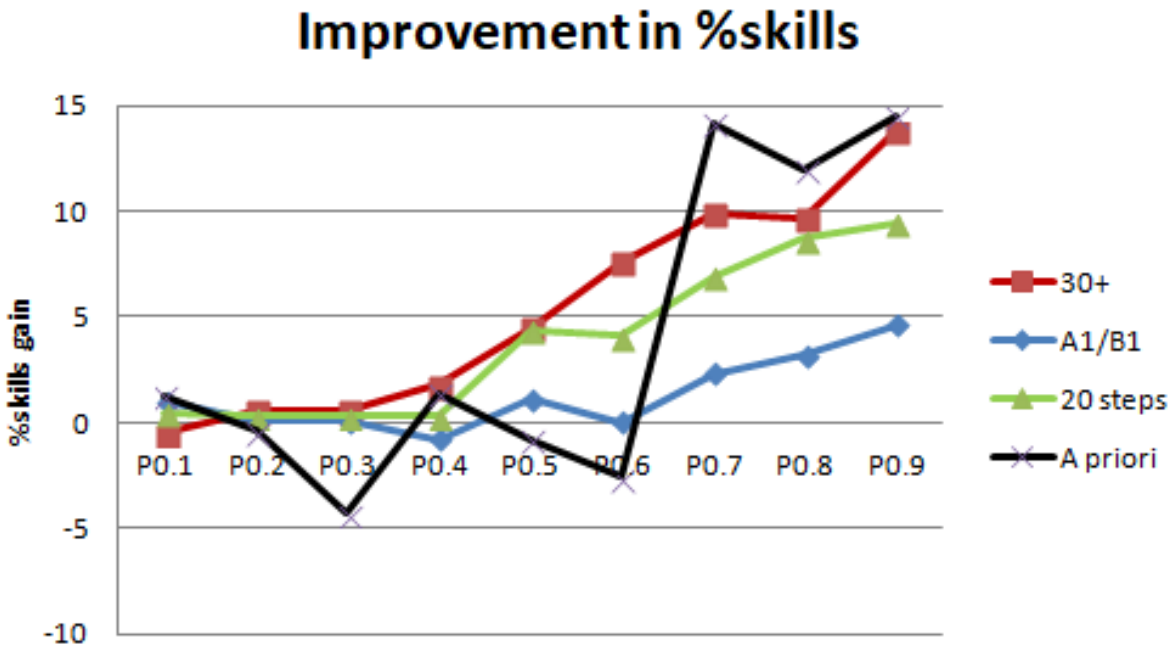

Figure 5.11: Gain in the average probability of knowing all skills by different conditions and a priori estimation in comparison to using average parameters from the start

leads to a period of time where suboptimal parameters are used, which has a particularly large impact in this case since the a priori method uses the same potentially optimal parameters from the beginning of the proposal of ZPD activities.

In order to obtain a better idea of how these methods behave on average, the mean and standard deviation for all methods was plotted using the values obtained for every student profile.

The results, visible in Figure 5.12, show that the ”30+” condition has both a higher mean and lower standard deviation on average than every other method, as well as showing that the average parameters have a high standard deviation value. This proves that the average parameters are quite suboptimal for most students. It is worth mentioning that every method ensured an average probability of knowing all skills of at least 90%, lower than the 97% threshold. This was expected since that threshold was defined for perfect classification of student profiles as opposed to this case’s occasionally wrong classification of random students which only use approximately optimal parameters even if classified correctly.

An additional comparison was also made using the mean and standard deviation for the average parameters, a priori and ”30+” methods, grouping the profile students into those with low ($P_{0.1}$ to $P_{0.3}$), medium ($P_{0.4}$ to $P_{0.6}$) and high ($P_{0.7}$ to $P_{0.9}$) learning speed.

The results are shown in Figure 5.13, and while the probability of knowing all skills are all very similar for low $baselearn$ students, the ”30+” method clearly has the highest probability and lowest standard deviation for the students classified as ”Medium”. For the high $baselearn$ group, however, the a priori

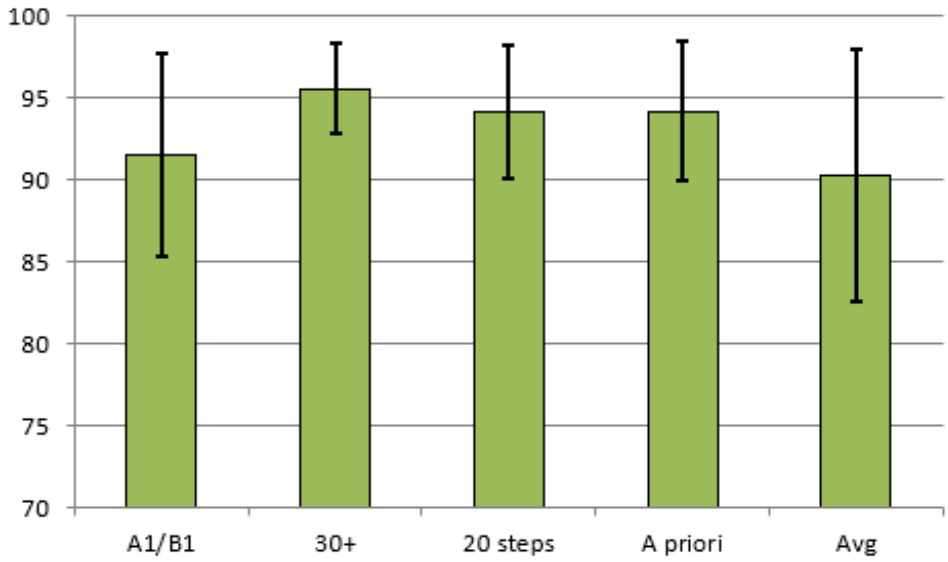

Figure 5.12: Mean and standard deviation for the average probability of knowing all skills for each reestimation condition

method shows a higher average probability of knowing all skills. This happens due to the previously explained reason that faster learning students spend less time in the ZPD, meaning that the relative number of time steps that they are using the optimal parameters obtained by methods which require a certain number of time steps for their condition to trigger is greater than it is for slower students.

After these tests, the statistical significance of the results was analyzed by performing t-tests on the a priori and ”30+” methods in comparison to using the average parameters. The data used was that of Table 5.3. Under the null hypotheses that there is no significant decrease in the average amount of time steps and that there is no significant improvement in the average probability of knowing all skills of these methods, the changes in terms of the number of time steps were not deemed statistically significant. However, the ”30+” condition showed a statistically significant improvement on the probability of knowing all skills for $p = 0.05$, with a p-value of approximately 0.04, allowing us to reject the null hypothesis regarding the lack of improvement in the probability of knowing all skills for this condition.

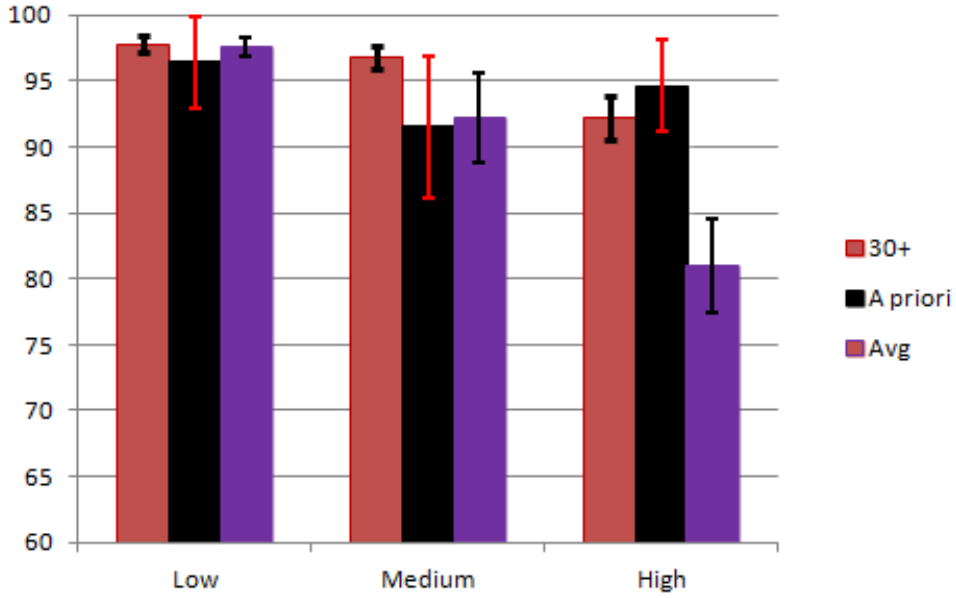

Figure 5.13: Comparison between the mean and standard deviation for the ”30+”condition, the a priori method and using the average parameters for students with different $baselearn$ values

6

Conclusion

Contents

6.1 Conclusions

This work shows that it is possible to group students through the dynamic assignment of approximately optimal parameters based on the estimation of each student's learning capabilities. This approach only requires the estimation of one attribute per student and the computation of the required algorithm is quick enough that it can be used as each students performs an exercise without issue while also ensuring a relatively low number of time steps and a high probability of learning every skill that is being taught. This ensures a better learning experience than individually estimating every model parameter per student since the parameter estimation is done in considerably less time steps than a brute force method would, while also guaranteeing a better balance between number of time steps and probability of learning all skills than simply using the same parameters for every student. Past data from the profile students is able to indirectly influence the learning experience of a new student due to having an impact on what model parameters this student uses, while data from other students is used to determine the initial GOZPDES parameters before the $baselearn$ estimation is performed.

The implemented solution for clustering - the student profiles - was made with this specific system in mind, and as such provides a simple solution that has no issue scaling with the number of students and which could easily be adapted into working with real students of different learning speeds. In theory, only the optimal parameters for each student profile would have to change.

6.2 System Limitations and Future Work

It is expectable that the behavior of real students cannot be fully captured within the attributes of the student class. The adaptation of the $baselearn$ attribute into reality is critical in allowing this model to be used with real students, along with the creation of an interface model. Since students may be better at certain exercise types and worse at others, the realism of the $baselearn$ attribute is dependent on the scope of the activities in the ZPD, as if the domain is sufficiently limited, this should not be an issue.

In the subject of the GOZPDES algorithm itself, since it is possible to preemptively adjust its parameters in order to fulfill certain thresholds in terms of number of time steps and probability of learning the taught skills, it may be interesting to adapt the model to allow the user to select a threshold for one of these measurements, with the program then automatically determining the optimal parameters for each student profile. This would allow the program to be used with different constraints, focusing on either teaching as much as possible within a limited time span or ensuring that the students learn every skill, taking a longer amount of time in order to do so.

Speaking of student profiles, an alternative clustering technique such as hierarchical clustering would have been interesting to attempt, as it would show an alternative approach to the problem.

Bibliography

[1] B. Clement, D. Roy, P. Oudeyer, and M. Lopes, "Multi-armed bandits for intelligent tutoring systems," *Journal of Educational Data Mining*, vol. 7, no. 2, pp. 1–22, 2013. [Online]. Available: http://arxiv.org/abs/1310.3174

[2] W. J. Clancey, "Intelligent Tutoring Systems: A tutorial Survey," p. 58, 1986.

[3] R. Freedman, S. Ali, and S. McRoy, "What is an intelligent tutoring system," *Intelligence*, vol. 11, no. 3, pp. 15–16, 2000. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.6543&rep=rep1&type=pdf

[4] K. R. Koedinger, J. R. Anderson, W. H. Hadley, M. A. Mark, J. R. Anderson, W. H. Hadley, and M. A. Mark, "Intelligent Tutoring Goes To School in the Big City," 1997.

[5] S. Schiaffino, P. Garcia, and A. Amandi, "eTeacher: Providing personalized assistance to e-learning students," *Computers and Education*, vol. 51, no. 4, pp. 1744–1754, 2008.

[6] S. P. Lajoie and A. Lesgold, "Apprenticeship training in the workplace: Computer-coached practice environment as a new form of apprenticeship," *Mach. Mediat. Learn.*, vol. 3, no. November, pp. 7–28, 1989. [Online]. Available: http://dl.acm.org/citation.cfm?id=78126.78128

[7] P. Pavlik, H. Cen, and K. Koedinger, "Performance Factors Analysis–A New Alternative to Knowledge Tracing," *Proceedings of the 2009 conference on Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling*, pp. 531–538, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1659529

[8] J. P. González-Brenes and Y. Huang, "General features in knowledge tracing: Applications to multiple subskills, temporal item response theory, and expert knowledge," *Proceedings of the 7th ...*, no. Edm, pp. 84–91, 2014. [Online]. Available: http://www.cs.cmu.edu/~joseg/files/fast_knowledge_tracing_with_features.pdf%5Cnpapers2://publication/uuid/62CFFC4D-3AFE-410C-9EA4-7CCC0B5BBCA0

[9] J. E. Beck and Xiaolu Xiong, “Limits to Accuracy: How Well Can We Do at Student Modeling?” *Proceedings of the 6th International Conference on Educational Data Mining*, 2013. [Online]. Available: http://www.educationaldatamining.org/EDM2013/papers/m_paper_04.pdf

[10] R. Nkambou, J. Bourdeau, and R. Mizoguchi, Eds., *Advances in Intelligent Tutoring Systems*, ser. Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, vol. 308. [Online]. Available: http://link.springer.com/10.1007/978-3-642-14363-2

[11] A. T. Corbett and J. R. Anderson, “Student Modeling and Mastery Learning in a Computer-Based Programming Tutor,” *Its ’92*, no. November, pp. 413–420, 1992.

[12] I. Padayachee, “Intelligent tutoring systems: Architecture and characteristics,” *Information Systems & Technology*, pp. 1–8, 2002. [Online]. Available: http://www.sacla.org.za/SACLA2002/Proceedings/Papers/Padayachee.pdf

[13] E. Altman, *Constrained Markov decision processes*. Chapman & Hall/CRC, 1999.

[14] S. R. Eddy, “Hidden Markov models,” *Current Opinion in Structural Biology*, vol. 6, no. 3, pp. 361–365, 1996.

[15] L. R. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,” pp. 257–286, 1989.

[16] L. R. Rabiner and B. H. Juang, “An introduction to hidden {M}arkov models,” *IEEE Signal Proc. Magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[17] R. Neal and G. Hinton, “A view of the EM algorithm that justifies incremental, sparse, and other variants,” *Learning in Graphical Models*, vol. 89, pp. 355–368, 1998.

[18] T. K. Moon, “The Expectation-Maximization Algorithm,” pp. 47–60, 1996.

[19] A. T. Corbett and J. R. Anderson, “Knowledge tracing: Modeling the acquisition of procedural knowledge,” pp. 253–278, 1994.

[20] B. V. D. Sande, “Properties of the Bayesian Knowledge Tracing Model,” *Journal of Educational Data Mining*, vol. 5, no. 2, pp. 1–10, 2013. [Online]. Available: http://gideon.eas.asu.edu/bvds/bayesian-knowledge-tracing.pdf

[21] J. E. Beck, K. M. Chang, J. Mostow, and A. Corbett, “Does help help? Introducing the bayesian evaluation and assessment methodology,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5091 LNCS, pp. 383–394, 2008.

[22] J. I. Lee and E. Brunskill, “The Impact on Individualizing Student Models on Necessary Practice Opportunities,” *Proceedings of the 5th International Conference on Educational Data Mining*, p. 8, 2012.

[23] Z. A. Pardos and N. T. Heffernan, “KT-IDEM: Introducing item difficulty to the knowledge tracing model,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6787 LNCS, pp. 243–254, 2011.

[24] Y. Xu and J. Mostow, “Using Logistic Regression to Trace Multiple Sub-skills in a Dynamic Bayes Net.” *Educational Data Mining*, pp. 241–246, 2011.

[25] H. Robbins, “Some Aspects of the Sequential Design of Experiments,” *Bulletin of the American Mathematical Society*, vol. 58, no. 5, pp. 527–535, 1952.

[26] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire, “The Non-Stochastic Multi-Armed Bandit Problem,” *SIAM Journal on Computing*, vol. 32, no. 1, pp. 48–77, 2003.

[27] L. Kaufman and P. J. Rousseeuw, *Finding groups in data : an introduction to cluster analysis*. Wiley-Interscience, 1990. [Online]. Available: https://books.google.pt/books?hl=pt-PT&lr=&id=YeFQHiikNo0C&oi=fnd&pg=PR11&dq=Finding+Groups+in+Data.&ots=5zw8z8KGtC&sig=8ThcoQSdt0q2ikPtDa_jPRwanOY&redir_esc=y#v=onepage&q&f=false

[28] R. Xu, “Survey of clustering algorithms for MANET,” *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005. [Online]. Available: http://arxiv.org/abs/0912.2303

[29] A. K. Jain, “Data clustering: 50 years beyond K-means,” *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2009.09.011

[30] A. K. Jain and R. C. Dubes, “Algorithms for Clustering Data,” p. 320, 1988. [Online]. Available: http://portal.acm.org/citation.cfm?id=SERIES10022.42779

[31] V. Garcia, E. Debreuve, and M. Barlaud, “Fast k nearest neighbor search using GPU,” *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW ’08. IEEE Computer Society Conference on*, pp. 1–6, 2008.

[32] S. A. Dudani, “The Distance-Weighted k-Nearest-Neighbor Rule,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-6, no. 4, pp. 325–327, 4 1976. [Online]. Available: http://ieeexplore.ieee.org/document/5408784/

[33] S. Engeser and F. Rheinberg, “Flow, performance and moderators of challenge-skill balance,” *Motivation and Emotion*, vol. 32, no. 3, pp. 158–172, 9 2008. [Online]. Available: http://link.springer.com/10.1007/s11031-008-9102-4

[34] H. Cen, K. Koedinger, and B. Junker, “Learning Factors Analysis – A General Method for Cognitive Model Evaluation and Improvement,” *Intelligent Tutoring Systems*, pp. 164–175, 2006. [Online]. Available: http://link.springer.com/chapter/10.1007/11774303_17

[35] A. Dempster, N. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *Journal of the Royal Statistical Society Series B Methodological*, vol. 39, no. 1, pp. 1–38, 1977. [Online]. Available: http://www.jstor.org/stable/10.2307/2984875

[36] T. Berg-kirkpatrick, A. Bouchard-Côté, J. DeNero, and D. Klein, “Painless unsupervised learning with features,” *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, no. June, pp. 582–590, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1858082

[37] K. R. Koedinger, A. T. Corbett, and C. Perfetti, “The Knowledge-Learning-Instruction Framework: Bridging the Science-Practice Chasm to Enhance Robust Student Learning,” *Cognitive Science*, vol. 36, no. 5, pp. 757–798, 7 2012. [Online]. Available: http://doi.wiley.com/10.1111/j.1551-6709.2012.01245.x

[38] C. Piech, J. Spencer, J. Huang, S. Ganguli, M. Sahami, L. Guibas, J. Sohl-dickstein, and K. Academy, “Deep Knowledge Tracing,” *arXiv*, pp. 1–13, 2015.

[39] X. Zhu, “Machine Teaching: An Inverse Problem to Machine Learning and an Approach Toward Optimal Education,” *Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 4083–4087, 2015. [Online]. Available: http://pages.cs.wisc.edu/~jerryzhu/machineteaching/pub/MachineTeachingAAAI15.pdf

[40] C. D. Lee, *Signifying in the zone of proximal development. An introduction to Vygotsky*, 2005.

[41] P.-Y. Oudeyer and F. Kaplan, “What is intrinsic motivation? A typology of computational approaches,” *Frontiers in Neurorobotics*, vol. 1, p. 6, 2007. [Online]. Available: http://journal.frontiersin.org/article/10.3389/neuro.12.006.2007/abstract

[42] B. Clement, P.-Y. Oudeyer, and M. Lopes, “A Comparison of Automatic Teaching Strategies for Heterogeneous Student Populations,” *Proceedings of the 9th International Conference on Educational Data Mining*, pp. 330–335, 2016.

[43] Y. Wang and N. Heffernan, “Extending knowledge tracing to allow partial credit: Using continuous versus binary nodes,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7926 LNAI, no. 2010, pp. 181–188, 2013.

[44] T. Barnes and J. Stamper, “Toward Automatic Hint Generation for Logic Proof Tutoring Using Historical Student Data.” [Online]. Available: http://link.springer.com/chapter/10.1007%2F978-3-540-69132-7_41

[45] S. Guha, N. Mishra, R. Motwani, and L. O’Callaghan, “Clustering data streams,” *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pp. 359–366, 2000. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=892124

[46] J.-M. Francois, “JAHMM: An implementation of hidden Markov models in Java,” 2010.

[47] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software,” *SIGKDD Explorations Newsletter*, vol. 11, no. 1, p. 10, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1656274.1656278

[48] S. Dasgupta, “Learning Polytrees,” *Uncertainty in Artificial Intelligence*, pp. 134–141, 1999.