Performance Distribution Tracing – Knowledge Tracing Using Probability Density Functions

Hildo Bijl

***Abstract*—The field of Knowledge Tracing is focused on predicting the success rate of a student for a given skill. Modern methods like Deep Knowledge Tracing provide accurate estimates given enough data, but being based on neural networks they struggle to explain how these estimates are formed. More classical methods like Bayesian Knowledge Tracing can do this, but they cannot give data on the accuracy of their estimates. As a result, with the current trend of requiring accountability from algorithms, these methods are found wanting.**

This paper presents a novel method, Performance Distribution Tracing (PDT), in which the distribution of the success rate is traced. This ensures that, for any estimate made, there is also data available on the accuracy of said estimate. In addition, by tracing distributions, it is possible to combine data from similar/related skills to come up with a more informed estimate of the success rate. This makes it possible to accurately predict exercise success rates, even when an exercise requires a combination of different skills to solve. That data can then be used to effectively recommend students which exercise to practice next.

An application to an engineering physics course has shown that the PDT algorithm works well in real-life settings. Students mainly appreciate the overview given of their learning process as well as the corresponding practice recommendations.

***Index Terms*—Education, Knowledge Tracing, Bayesian Methods, Distribution Tracing, Machine Learning**

I. Introduction

WHEN a student studies a course in a traditional classroom, the teacher tracks his/her progress, often even subconsciously. Every interaction between student and teacher influences the internal idea the teacher has of the student. 'What skills have been mastered? And which are still lacking?' This determines the way the teacher tutors the student.

With the advent of digital tutoring systems, attempts have been made to automate this process. Most work relates to *Knowledge Components* (KCs), as described by [1]. A distinction can be made between *explicit KCs* (facts/principles) and *implicit KCs* (skills). Often skills are practiced and tested through exercises, and the fundamental question then is, 'What is the chance that the student will do the next exercise correctly?' This probability is known as the *success rate* and the study of tracking/estimating it is known as *Knowledge Tracing* (KT). A good overview of this field is given by [2] though a brief summary is provided below.

Classical work on KT revolves around *Bayesian Knowledge Tracing* (BKT) [3]. This method directly estimates the success rate, constantly updating this estimate on each subsequent exercise execution using a hidden Markov model. Probabilities of the student slipping (failing despite having mastery) or guessing (succeeding while not having mastery) are taken into account [4] as well as potentially the exercise difficulty [5]. Extensions exist for further individualization towards students, as outlined by [6], [7].

Instead of using probabilistic models, an alternative set of methods uses logistic models. The fundamentals have been presented by [8] in the *Learning Factors Analysis* (LFA) method. By using weight factors to account for the difficulty of a KC, the ease of learning of a KC and various other parameters, and by plugging the result into a logistic function, the success rate for a KC is estimated. Extensions/variants include *Performance Factor Analysis* (PFA) [9] and *Knowledge Tracing Machines* (KTM) [10].

The methods above mainly work for individual KCs, struggling to take into account links between KCs. Often data on these links – the so-called *domain knowledge* – is available. The *Dynamic Bayesian Knowledge Tracing* (DBKT) method [11] extends on BKT to incorporate these links.

With the introduction and growing popularity of *Massive Open Online Courses* (MOOCs) [12] the amount of data that can be used for KT has increased significantly. As a result, more advanced methods like *Deep Knowledge Tracing* (DKT) have appeared, with fundamental work done by [13]. In DKT recurrent neural networks are used to model the student performance as he/she works through a course. Though this method requires a large amount of data, and is prone to overfitting [14], it does not require domain knowledge: the algorithm determines for itself which links between KCs exist. Possible extensions include taking into account the learning curve [15].

Recent trends in machine learning put more importance on accountability [16]. Whenever an algorithm makes a prediction/estimate, several questions rise up.

- What did the algorithm base this prediction on?
- How does this prediction relate to the domain knowledge?
- How certain can we be of the given estimate?

Ideally all these questions can be answered. Given these requirements, we see that DKT actually performs rather badly: it is a black-box method that struggles to explain its predictions and does not apply any currently existing domain knowledge. BKT does better: its simplicity allows it to explain its estimates. However, it does not have any information stored about the certainty of its estimates. As a result, the need for an alternative method arises, one which does track the certainty of its estimates.

Hildo Bijl (hildo.bijl@hu.nl) is with the Utrecht University of Applied Sciences, Utrecht, The Netherlands.

Manuscript received December 15, 2021.

The result is a novel method: *Performance Distribution Tracing* (PDT). In this method, the success rate is not estimated as a number, but instead as a random variable whose distribution is updated after each new observation. For the mathematical background on random variables, see for instance [17]. Tracing the distribution provides not only data on the accuracy of every estimate, but it also allows for the combining of data from multiple sources – for instance multiple KCs/skills – to provide more informed estimates. This allows PDT to efficiently take into account domain knowledge to improve its estimates.

This paper is set up as follows. Chapter II studies tracing the performance of a student for a single skill. In Chapter III these methods are extended to take into account exercises involving multiple skills. This is further extended in Chapter IV, studying links between various skills and how these links can be taken into account to get more informed success rate estimates. The PDT method has been incorporated into a tutoring app that has been applied to a basic engineering course. Results of this are discussed in Chapter V. The paper closes off with conclusions and recommendations in Chapter VI.

II. Modeling the proficiency of a single skill

Consider any single skill A (an implicit knowledge component) that a student might do. This skill A could be as basic as 'multiply two small numbers' or as advanced as 'perform a complex engineering mechanics calculation'. For this skill, we want to track the student's performance, update this when provided with new data, and make predictions on future executions.

A. Mathematically describing the probability of success

Key to modeling the student's proficiency at skill A is the *success rate*: the chance a that a student performs it successfully. Existing KT methods see a as a deterministic variable that must be found. However, this provides no data on the confidence with which a is known: is it a rough guess or a near-certain estimate?

This problem is solved if we describe the success rate by a random variable $\underline{a}$. (The underline notation denotes random variables.) In this case $\underline{a}$ is not described by a single value, but by its distribution, expressed through its Probability Density Function (PDF) $f_{\underline{a}}(a)$. Examples of performance PDFs are shown in Figure 1. Note that $\underline{a}$ only takes values between 0 and 1 because it is a probability. We hence always have $f_{\underline{a}}(a) = 0$ for $a < 0$ or $a > 1$.

To describe the PDF of a success rate $\underline{a}$, it is convenient to use basis functions. Specifically, we will write $f_{\underline{a}}(a)$ as

$$f_{\underline{a}}(a) = \sum_{i=0}^{n} c_i (n+1) \binom{n}{i} a^i (1-a)^{n-i}; \tag{1}$$

for some *order* n and some set of coefficients c_i, with $0 \quad i \quad n$. Alternatively, we can define the basis functions $g_{i;n}(a)$ for some order n as

$$g_{i;n}(a) = (n+1) \binom{n}{i} a^i (1-a)^{n-i}; \tag{2}$$

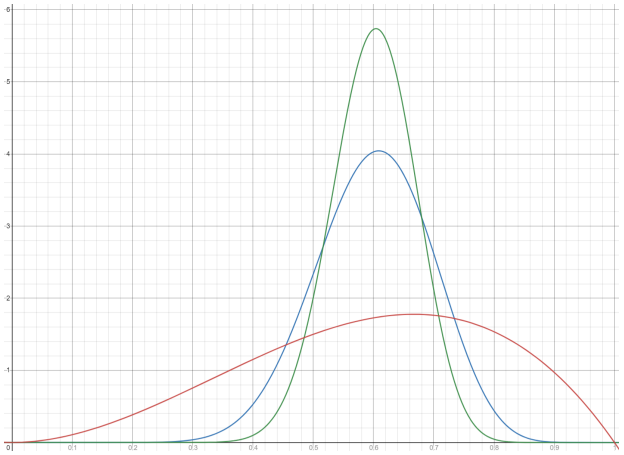

Fig. 1. Three example distributions $f_{\underline{a}}(a)$ plotted from 0 to 1. They all have an expected value of 0.6, denoting a 60% estimated success rate, but their peakedness and hence their degree of certainty varies.

and subsequently write the PDF of $\underline{a}$ as

$$f_{\underline{a}}(a) = \sum_{i=0}^{n} c_i g_{i;n}(a) = \mathbf{c}^T \mathbf{g_n}(a). \tag{3}$$

Note that the *coefficient vector* $\mathbf{c}$ is the vector with all coefficients c_i and identically the *basis function vector* $\mathbf{g_n}(a)$ is the vector function containing all basis functions $g_{i;n}(a)$. Obviously, not all functions can be described through this basis form, but the PDFs we will encounter generally do, and this allows us to reduce a complete function to a simple set of coefficients.

There are some interesting consequences to describing the distribution of $\underline{a}$ in this way. Because $f_{\underline{a}}(a)$ is a PDF, its integral must equal one. Hence,

$$\int_0^1 f_{\underline{a}}(a)\,da = \sum_{i=0}^{n} c_i \int_0^1 g_{i;n}(a)\,da = \sum_{i=0}^{n} c_i = 1. \tag{4}$$

Here we have used the fact that the integral over a basis function $g_{i;n}(a)$ always equals one, as proven in Appendix A-A. The above shows that the sum of all the coefficients must always equal one. If this is the case, the coefficients are said to be *normalized*. If the coefficients are not normalized, then it is always possible to normalize them by dividing each coefficient by the sum of all coefficients. In practice this is often done, as well as ensure that all coefficients remain positive, to compensate for potential numerical inaccuracies in computers.

Given $f_{\underline{a}}(a)$ we can also calculate the expected probability of success. This is given by

$$\mathrm{E}[\underline{a}] = \sum_{i=0}^{n} c_i \int_0^1 a g_{i;n}(a)\,da = \sum_{i=0}^{n} c_i \frac{i+1}{n+2}. \tag{5}$$

More generally, any moment $\mathrm{E}[\underline{a}^m]$ can be calculated through

$$\mathrm{E}[\underline{a}^m] = \sum_{i=0}^{n} c_i \frac{(n+1)!}{(n+m+1)!} \frac{(i+m)!}{i!}. \tag{6}$$

Instead of looking at the success rate $\underline{a}$, we can also study the *failure rate* $1 \quad \underline{a}$. The distribution of the failure rate,

described by $f_{1-\underline{a}}(a)$, can be found by reversing the order of the coefficients c_i. That is,

$$f_{1-\underline{a}}(a) = \sum_{i=0} c_{n-i} g_{i;n}(a). \quad (7)$$

This concludes all the relevant properties of our basis function description. It is time to apply it.

B. Updating the skill performance distribution

Initially, when no data is present on a skill A, we start with the flat prior $f_{\underline{a}}(a) = 1$. This corresponds to a coefficient vector of $\mathbf{c} = [1]$. The order of this coefficient vector is hence 0: there is no information yet.

Next suppose that, after a student has done $k-1$ exercises for some skill A, the performance distribution $f_{\underline{a}}(a|D_{k-1})$ is known. Here, D_{k-1} denotes all data related to the first $k-1$ exercise executions. In other words, the coefficient vector $\mathbf{c}$ is known up to this point. If the student then performs an exercise again (execution k), which is either a success or a failure, how is this data incorporated?

The posterior distribution is given through Bayes' law as

$$f_{\underline{a}}(a|D_k) = \frac{p(D_k|a; D_{k-1}) f_{\underline{a}}(a|D_{k-1})}{p(D_k|D_{k-1})}. \quad (8)$$

Note that, if the last exercise execution was successful, then $p(D_k|a; D_{k-1}) = a$, while on a failure it equals $p(D_k|a; D_{k-1}) = 1-a$. Also note that $p(D_k|D_{k-1})$ is a constant. Assuming that afterwards we normalize our coefficient vector $\mathbf{c}$, which we always do, we can safely ignore it.

The posterior distribution $f_{\underline{a}}(a|D_k)$ described above is once more a performance distribution in basis form. That means we can describe it through a new set of coefficients. If the old distribution has order n, the new distribution will have order $n_* = n+1$ and its coefficients are (for $0 \le i \le n_*$)

$$c_i^* = \begin{cases} i c_{i-1} & \text{on success;} \\ (n+1-i) c_i & \text{on failing;} \end{cases} \quad (9)$$

where afterwards we apply normalization to the coefficients to get their sum to equal one once more. Note that $\mathbf{c}^*$ (the star superscript) denotes the coefficients of the updated distribution, incorporating the latest observation. Through this update law, we can continuously incorporate more data, allowing the performance distribution to become more peaky over time.

C. Inferring the success rate for the next execution

The above has assumed that the actual value of the success rate $\underline{a}$ is a constant: it may be unknown, but it does not vary over time. In reality this is not the case. After every exercise the student may have learned something new, or possibly misinterpreted something. The value of $\underline{a}$ hence slowly shifts, upwards or downwards. This is known as the *learning effect* [18].

To account for the learning effect, we use multiple random variables: we define $\underline{a}_k$ as the probability of success of execution k of skill A. Initially we know (as our prior) the distribution $f_{\underline{a}_1}(a)$. After the first exercise, this will be updated using (9) as $f_{\underline{a}_1}(a|D_1)$. But what does this tell us about the probability of success $\underline{a}_2$ of the second execution?

To be able to say anything about $\underline{a}_2$, we must first assume a joint prior between subsequent skill executions $\underline{a}_k$ and $\underline{a}_{k+1}$: how similar are their success rates? A joint prior that is both descriptive and mathematically convenient is

$$f_{\underline{a}_k;\underline{a}_{k+1}}(a_k; a_{k+1}) = \frac{\mathbf{g}_{\mathbf{n_s}}^T(a_k)\mathbf{g}_{\mathbf{n_s}}(a_{k+1})}{n_s+1}; \quad (10)$$

where n_s is known as the *smoothing order*. The shape of this prior is shown in Figure 2.

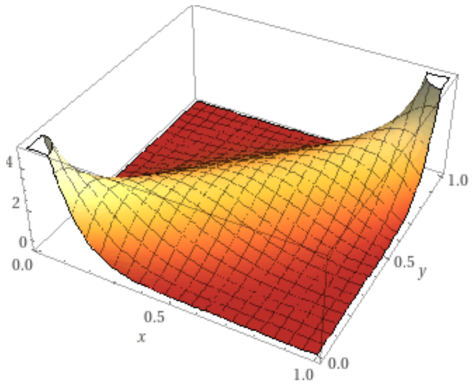

Fig. 2. The joint prior for two subsequent skill success rates $\underline{a}_k$ and $\underline{a}_{k+1}$, for smoothing order $n_s = 10$. Note that, given a value of $\underline{a}_k$, the value of $\underline{a}_{k+1}$ will very likely be similar. Higher smoothing orders n_s give a more peaked ridge at the diagonal line, and hence assume a stronger correlation between $\underline{a}_k$ and $\underline{a}_{k+1}$.

The above prior can be used to infer the distribution of $\underline{a}_{k+1}$ from the distribution of $\underline{a}_k$. For this, we must first use the definition of the conditional distribution to write

$$f_{\underline{a}_k;\underline{a}_{k+1}}(a_k; a_{k+1}|D_k) = f_{\underline{a}_{k+1}}(a_{k+1}|a_k; D_k) f_{\underline{a}_k}(a_k|D_k). \quad (11)$$

Note that, given $\underline{a}_k$, the data D_k does not add any information to $\underline{a}_{k+1}$, so it can be omitted. Applying the conditional distribution in reverse, and marginalizing over $\underline{a}_k$, we find

$$f_{\underline{a}_{k+1}}(a_{k+1}|D_k) = \int_0^1 \frac{f_{\underline{a}_k;\underline{a}_{k+1}}(a_k; a_{k+1}) f_{\underline{a}_k}(a_k|D_k)}{f_{\underline{a}_k}(a_k)} da_k. \quad (12)$$

The resulting distribution is once more one in basis form. We can hence solve the integral and derive the resulting coefficients $c_{k+1;i}$. For the mathematically curious, the full derivation is given in Appendix A. We only present the final outcome here as

$$c_{k+1;i} = \sum_{j=0}^{n_*} \frac{\binom{i+j}{i}\binom{n_*+n_s-i-j}{n_*-j}}{\binom{n_*+n_s}{n_*}} c_{k;j}^*; \quad (13)$$

where afterwards the coefficients must be normalized as usual. Note that the coefficients $c_{k;j}^*$ describe $f_{\underline{a}_k}(a_k|D_k)$ (the success rate of execution k *given* the data on the result of execution k) while $c_{k+1;i}$ describe $f_{\underline{a}_{k+1}}(a_{k+1}|D_k)$ (the success rate of execution $k+1$ *without* knowing the result of execution $k+1$). Also note that the order of this new set of coefficients $c_{k+1;i}$ is always the smoothing order n_s.

When applying the joint prior in this way, the distribution of $\underline{a}_{k+1}$ is always slightly flatter (less peaked) than the distribution of $\underline{a}_k$. As a result, this step is known as the *smoothing step*. By applying this smoothing, we effectively incorporate the uncertainty of the learning effect. The smaller n_s is, the more smoothing is applied, while a large value of n_s leaves the distribution nearly unaffected.

The only unanswered question is how to choose n_s. In practice this is done based on various settings. The full method is discussed in Appendix B but it can be summarized in a few simple rules. If a student has practiced the skill a lot, then the learning effect is smaller so n_s is chosen to be larger to have less smoothing. Similarly, if the time since the last skill execution is large, the student may have forgotten a lot, so n_s is chosen to be smaller to have more smoothing. Because the smoothing step is time-dependent, the coefficients that are stored in the database for each student are always the coefficients $\mathbf{c}_\mathbf{k}^*$ *after* incorporating new observations yet *prior* to smoothing.

III. Handling exercises combining multiple skills

So far we have considered exercises in which only a single skill needed to be applied. In practice, many exercises require a variety of skills. How can we incorporate results of these exercises into the corresponding skill performance distributions? And how do we use this to predict exercise success rates?

A. Mathematically defining the exercise set-up

The first step is to describe the exercise. Consider an exercise X requiring skills A and B. The exact way this exercise is set up can be described through various operators.

- The and-operator: if a successful execution of the exercise requires a successful execution of both A and B, then we write $X = \text{and}(A;B)$. The chance $\underline{x}$ that a student does both A and B correctly is $\underline{x} = \underline{a}\underline{b}$.
- The or-operator: if an exercise can be solved by either doing A or B, with each method being suitable, then we write $X = \text{or}(A;B)$. In this case $\underline{x} = 1 - (1-\underline{a})(1-\underline{b}) = \underline{a} + \underline{b} - \underline{a}\underline{b}$. Note that $\underline{x} \geq \max(\underline{a};\underline{b})$ because a student can use the other method/skill to check his/her answer.

In theory an exercise can have any kind of set-up. For instance, we may have $X = \text{and}(A;\text{or}(A;B))$, in which case $\underline{x}(\underline{a};\underline{b}) = \underline{a}^2 + \underline{a}\underline{b} - \underline{a}^2\underline{b}$. The first expression '$\text{and}(A;\text{or}(A;B))$' is known as the *exercise set-up* while the second expression '$\underline{a}^2 + \underline{a}\underline{b} - \underline{a}^2\underline{b}$' is the corresponding *probability polynomial*. Note that any set-up can be readily turned into a probability polynomial.

B. Updating the skill performance distributions

Suppose that a student does exercise X correctly, or alternatively he/she fails it. This then provides another data point D_X that can be used to update the distributions of both $\underline{a}$ and $\underline{b}$. We consider how to update the distribution of $\underline{a}$. It works identically for $\underline{b}$, as well as any other potentially involved skill.

Our goal is to find the posterior $f_{\underline{a}}(a|D_X)$. (All previously known data D_A and D_B on skills A and B is also taken into account, but this is left out of the notation for brevity.) By marginalizing the joint distribution $f_{\underline{a};\underline{b}}(a;b|D_X)$ with respect to $\underline{b}$, followed up with Bayes' law, we find

$$f_{\underline{a}}(a|D_X) = \int_0^1 \frac{p(D_X|a;b)\, f_{\underline{a};\underline{b}}(a;b)}{p(D_X)}\, db. \tag{14}$$

Note that $p(D_X)$ is a constant, not depending on $\underline{a}$, so we can leave it out if we normalize coefficients afterwards. Also note that $p(D_X|a;b)$ follows from the probability polynomial $x(a;b)$ as

$$p(D_X|a;b) = \begin{cases} x(a;b) & \text{on success;} \\ 1 - x(a;b) & \text{on failing.} \end{cases} \tag{15}$$

If we also assume independence of $\underline{a}$ and $\underline{b}$ then we can write

$$f_{\underline{a}}(a|D_X) \propto \int_0^1 p(D_X|a;b)\, f_{\underline{b}}(b)\, db \quad f_{\underline{a}}(a) = h_{\underline{a}}(a)\, f_{\underline{a}}(a); \tag{16}$$

where we have defined $h_{\underline{a}}(a)$ as the part between brackets in the above equation. Note that $h_{\underline{a}}(a)$ is merely the expected value, with respect to all random variables other than a, of (on success) the probability polynomial $x(a;b)$ or (upon failure) the inverse $1 - x(a;b)$.

$h_{\underline{a}}(a)$ is a polynomial that only depends on a. This means we can write it as

$$h_{\underline{a}}(a) = \sum_{i=0}^{n_p} k_i a^i; \tag{17}$$

for some polynomial order n_p and some set of constants $k_0;\ldots;k_{n_p}$. The exact value of these constants can be found from the probability polynomial with the help of (6).

Before we continue, we must first put $h_{\underline{a}}(a)$ in an alternate form. We want to write $h_{\underline{a}}(a)$ as

$$h_{\underline{a}}(a) = \sum_{i=0}^{n_p} k'_i \binom{n_p}{i} a^i (1-a)^{n_p - i}; \tag{18}$$

for another set of constants $k'_0;\ldots;k'_{n_p}$. The link between these two sets of constants is

$$\binom{n_p}{n_p - i} k'_i = \sum_{j=0}^{n_p} \binom{n_p - j}{n_p - i} k_j. \tag{19}$$

Thanks to this alternate way of writing $h_{\underline{a}}(a)$, we can now update the coefficients of $f_{\underline{a}}(a)$. If this PDF used to have order n, with coefficients c_i for $0 \leq i \leq n$, then the posterior PDF $f_{\underline{a}}(a|D_X)$ will be of order $n_* = n + n_p$. Its coefficients c_i^*, with $0 \leq i \leq n_*$, can be found through

$$c_i^* = \sum_{j=\max(0;i-n)}^{\min(n_p;i)} \binom{i}{j} \binom{n_* - i}{n_p - j} c_{i-j} k'_j; \tag{20}$$

where afterwards coefficients must once more be normalized. Note that this method is a generalization of the update law of (9). It allows us to update the distributions of any skills A, B, and potentially more, based on exercises with any set-up involving these skills.

In practice this update law has some intuitive consequences. Consider the situation where a student has mostly mastered

skill A and is still struggling with skill B. If he/she then tries an exercise with set-up $\mathsf{and}(A;B)$ and fails, then the probability theory inherent in the above method automatically searches for 'blame'. In this case, the fault most likely lies in skill B, so that skill is more strongly penalized, while the success rate for skill A is hardly adjusted downwards. This behavior is exactly what seems sensible, showing that the system works in the desired way.

C. Inferring the success rate for the exercise

When considering which exercise to present to the student, it is useful to know how likely the student is to successfully complete each exercise. Given an exercise X with known set-up and probability polynomial, we want to know the success rate $\underline{x}$, or more simply put the expected value $\mathsf{E}[\underline{x}]$ of it.

The expected value $\mathsf{E}[\underline{x}]$ is actually straightforward to find. If we assume that $\underline{a}$ and $\underline{b}$ are independent, then the expected value of the probability polynomial $\mathsf{E}[\underline{x}(\underline{a};\underline{b})]$ follows directly from the application of (6).

In some cases it is also useful to know the complete distribution of $\underline{x}$. This tells us (among others) how certain we are of the estimated success rate $\mathsf{E}[\underline{x}]$. Finding the distribution of $\underline{x}$ is rather involved, mostly because it generally *cannot* be described through our basis functions. However, we can define a nearly identical random variable $\underline{\hat{x}}$ whose distribution *can* be described through basis functions.

We want the exercise success rate $\underline{x}$ and this new random variable $\underline{\hat{x}}$ to have as similar distributions as possible. To capture this similarity, we define the prior $f_{\underline{\hat{x}};\underline{x}}(\hat{x};x)$, identically to (10), as

$$f_{\underline{\hat{x}};\underline{x}}(\hat{x};x) = \frac{\mathbf{g}_{\mathbf{n_i}}^T(\hat{x})\mathbf{g}_{\mathbf{n_i}}(x)}{n_i+1}; \tag{21}$$

for some *inference smoothing order* n_i. In practice, the order n_i for this application is often chosen to be on the lower side ($n_i \quad 10$) for practical reasons we will soon see.

We want to find the posterior distribution $f_{\underline{\hat{x}}}(\hat{x}|D_{A;B})$ given all data $D_{A;B}$ on skills A and B. Identically to how we found (12) we can find

$$f_{\underline{\hat{x}}}(\hat{x}|D_{A;B}) = \int_0^1 \frac{f_{\underline{\hat{x}};\underline{x}}(\hat{x};x) f_{\underline{x}}(x|D_{A;B})}{f_{\underline{x}}(x)}\,dx. \tag{22}$$

This integral can subsequently be solved and rewritten into the basis function form. This gives us a set of coefficients $c_{x;i}$ describing the posterior distribution of $\underline{\hat{x}}$. These coefficients satisfy

$$c_{x;i} = \int_0^1\int_0^1 g_{i;n_i}(x(a;b)) f_{\underline{a}}(a|D_A) f_{\underline{b}}(b|D_B)\,da\,db; \tag{23}$$

where subsequent normalization must be applied, as usual. Alternatively, we can say that

$$\mathbf{c_x} = \mathsf{E}[\mathbf{g}_{\mathbf{n_i}}(\underline{x}(\underline{a};\underline{b}))]. \tag{24}$$

So the coefficients $c_{x;i}$ are simply the expected values of the basis functions $g_{i;n_i}$ when inserting the probability polynomial into these basis functions. This idea works identically if more than two skills A and B are involved, and it is known as the *inference step* of the algorithm.

The remaining question is how to calculate these coefficients from the coefficients of $\underline{a}$ and $\underline{b}$. Note here that each basis function $g_{i;n_i}$ is also a polynomial function. Inserting a polynomial into a polynomial function once more gives a polynomial. If $g_{i;n_i}$ does not have too large powers (that is, we keep n_i reasonably small) this can be expanded through a binomial expansion. Subsequently, the expected value can be found through (6).

Finally, it might be interesting to discuss the intuitive view of this new variable $\underline{\hat{x}}$: how can we interpret it? Some may argue that $\underline{x}$, which follows directly from $\underline{a}$ and $\underline{b}$ through the probability polynomial $\underline{x}(\underline{a};\underline{b})$, is *not* the most accurate way to describe whether the student will successfully complete exercise X. After all, when doing exercise X, the student must also recognize the steps to take, which has nothing to do with A and B individually, but does affect the success rate for exercise X. As a result, we must define a *true success rate* $\underline{\hat{x}}$, which takes the estimated success rate $\underline{x}$ based on the skills A and B, and adds some uncertainty (read: smoothing of the distribution) on top of this.

IV. Describing relations between skills

The strength of the PDT algorithm lies in its ability to link related skills. A course generally does not consist of a collection of unrelated skills. The skills build up on each other in an often complex way. This can be modeled and incorporated. The complete method may seem convoluted at first, but the overview at the end (Figure 3) should clarify this.

A. Defining the skill set-up

A *composite skill* is a skill made up of multiple smaller *subskills*. For instance, to calculate $2 + 3 \quad 4$ you must first figure out the order of operations, then apply multiplication and finally apply addition. In this simple example, we could say that skill S (evaluating basic expressions) has a set-up of $\mathsf{and}(A;B;C)$, with A, B and C the aforementioned subskills.

Skills have a set-up similar to exercises, but while exercises always have a *deterministic set-up*, only using and and or operations, skills may have a *non-deterministic set-up*. In the above example, instead of using addition, we may perhaps require subtraction half the times, resulting in a varying set-up. We therefore add the following non-deterministic operations.

- The pick-operator: from a list of subskills, we select only one. For instance, if a skill first requires either A or B (but always only one of them) followed by C, then we write $S = \mathsf{and}(\mathsf{pick}(A;B);C)$. To find the probability polynomial, we may reduce $\mathsf{pick}(A;B)$ to $\frac{1}{2}\underline{a} + \frac{1}{2}\underline{b}$. Extensions exist where multiple skills from a list are selected and/or weights are applied upon selection.
- The part-operator: if a skill only requires a skill A in a part p of the cases, always followed by a skill B, then we may write $S = \mathsf{and}(\mathsf{part}(A;p);B)$. Alternatively, if sometimes we can apply either A or B, and sometimes only B, we may write $S = \mathsf{or}(\mathsf{part}(A;p);B)$.
  For the part-operator, the probability polynomial depends on the surrounding operator. On a surrounding and-operator we reduce $\mathsf{part}(A;p)$ to $1 \quad p(1 \quad \underline{a})$, while on a surrounding or-operator $\mathsf{part}(A;p)$ becomes $p\underline{a}$.

With these extra operators, it is still always possible to turn a skill set-up into a probability polynomial. As a result, using the merging methods from Section III-C, we can always predict the success rate of a skill S based on data from its subskills, just like for an exercise X.

B. Merging observations on skills and subskills

Consider the situation where a composite skill S has subskills A and B, and where data is available on *all* these skills.

- First the student practices subskills A and B, giving data $D_{A;B}$. Using the methods from Section III-C – specifically the coefficients from (24) – we can hence infer $f_{\underline{s}}(s|D_{A;B})$.
- Subsequently the student practices skill S directly, giving data D_S. Using *only* this data, we can also find a distribution $f_{\underline{s}}(s|D_S)$. This is done through (9) or more generally with (20).

How can this data then be 'merged' into $f_{\underline{s}}(s|D_{A;B};D_S)$? To answer this question, we must apply Bayes' law,

$$f_{\underline{s}}(s|D_{A;B};D_S) = \frac{p(D_{A;B};D_S|s)f_{\underline{s}}(s)}{p(D_{A;B};D_S)}. \tag{25}$$

Assuming that the observation sets $D_{A;B}$ and D_S are conditionally independent given $\underline{s}$, we may reduce $p(D_{A;B};D_S|s)$ to $p(D_{A;B}|s)p(D_S|s)$. Applying Bayes' law in the reverse direction subsequently gives

$$f_{\underline{s}}(s|D_{A;B};D_S) = \frac{p(D_{A;B})p(D_S)}{p(D_{A;B};D_S)}\frac{f_{\underline{s}}(s|D_{A;B})f_{\underline{s}}(s|D_S)}{f_{\underline{s}}(s)}. \tag{26}$$

Note that all terms apart from $f_{\underline{s}}(s|D_{A;B})$ and $f_{\underline{s}}(s|D_S)$ are constants. We can therefore find the posterior distribution of $\underline{s}$ simply by multiplying these separately inferred distributions and normalizing the result.

Let's say that $f_{\underline{s}}(s|D_{A;B})$ has order $n_{A;B}$ and coefficients $c_i^{A;B}$, and identically $f_{\underline{s}}(s|D_S)$ has order n_S and coefficients c_i^S. The resulting posterior distribution $f_{\underline{s}}(s|D_{A;B};D_S)$ then has order $n_m = n_{A;B} + n_S$ and coefficients c_i^m satisfying

$$c_i^m = \sum_{j=\max(0,i-n_{A;B})}^{\min(n_S,i)} \binom{i}{j}\binom{n_m}{n_S}^{-1}\binom{i}{j} c_{i-j}^{A;B} c_j^S; \tag{27}$$

where the coefficients must be normalized afterwards. Note the similarity of the above inference law with the update law (20).

The above method effectively shows how to merge two distributions of $f_{\underline{s}}(s)$, based on different, separate and conditionally independent data, together into one distribution. As a result the above method is known as *merging distributions*, which is also why we use a subscript m here.

C. Incorporating correlations between skills

Next to parent/child dependencies between skills, we in practice also encounter correlated skills. For instance, the skill 'expand brackets with numbers' with example exercise 'expand $(2 + 3)\ \ 4$' is generally strongly correlated with the skill 'expand brackets with letters' with example exercise 'expand $(a + b)\ \ c$'. Note that these skills are similar but not identical: many a mathematics teacher can confirm that students who have mastered the first skill often still struggle with the second one. Nevertheless, someone's performance on the first skill *does* give a little bit of information about the expected performance on the second skill.

To model this, we consider a skill S with a correlated skill R. We write their success rates as $\underline{s}$ and $\underline{r}$, respectively. We now want to use data D_R on skill R to infer the distribution of $\underline{s}$. Once more, the first step is to define a joint prior between $\underline{r}$ and $\underline{s}$. Identically to (10) we assume

$$f_{\underline{r};\underline{s}}(r;s) = \frac{\mathbf{g}_{\mathbf{n_c}}^T(r)\mathbf{g}_{\mathbf{n_c}}(s)}{n_c+1}; \tag{28}$$

for some *correlation smoothing order* n_c. The stronger the correlation, the higher n_c must be. In practice correlations are often weak, so n_c does not go above a value of 10.

Once the joint prior is defined, we notice that the problem we have here is exactly the same as that from Section II-C. The distribution $f_{\underline{s}}(s|D_R)$ is hence simply the smoothed version of $f_{\underline{r}}(r|D_R)$, using smoothing order n_c. The coefficients follow from (13).

Once $f_{\underline{s}}(s|D_R)$ has been determined, we can merge it into $f_{\underline{s}}(s|D_S)$ using the methods from Section IV-B; specifically through (27). This gives us the posterior $f_{\underline{s}}(s|D_R;D_S)$. After all, the problem is identical to when we tried to derive the posterior distribution of $\underline{s}$ based on data from subskills. Additionally, in case data from subskills is *also* present, we can merge that in too. The order of merging does not matter.

A more complex problem appears when there are groups of correlated skills. Imagine there are two skills Q and R that are both correlated with skill S. You could treat all these correlations individually and separately merge $f_{\underline{s}}(s|D_Q)$ and $f_{\underline{s}}(s|D_R)$ into $f_{\underline{s}}(s|D_S)$. However, this neglects correlations between skills Q and R which are most likely present. To take this into account, we can define a joint prior like

$$f_{\underline{q};\underline{r};\underline{s}}(q;r;s) = \frac{1}{n_c+1}\sum_{i=0}^{n_c} g_{i;n_c}(q)g_{i;n_c}(r)g_{i;n_c}(s). \tag{29}$$

By applying Bayes' law back and forth a few times, similar to what we did for (26), we can find that the joint posterior distribution given data on Q and R is proportional to

$$f_{\underline{q};\underline{r};\underline{s}}(q;r;s|D_Q;D_R) \quad f_{\underline{q}}(q|D_Q)f_{\underline{r}}(r|D_r)f_{\underline{q};\underline{r};\underline{s}}(q;r;s). \tag{30}$$

Marginalizing over $\underline{q}$ and $\underline{r}$ then results in the posterior distribution $f_{\underline{s}}(s|D_{Q;R})$. The lengthy mathematics are omitted for reasons of brevity, but the final procedure is as follows.

- Start with the coefficients c_i^q and c_i^r describing $f_{\underline{q}}(q|D_Q)$ and $f_{\underline{r}}(r|D_R)$. (Ensure that the learning effect and time decay are already taken into account.)
- Smooth these distributions separately with (13) giving coefficients $c_i^{q'}$ and $c_i^{r'}$ describing $f_{\underline{s}}(s|D_Q)$ and $f_{\underline{s}}(s|D_R)$.
- Multiply the coefficients $c_i^{q'}$ and $c_i^{r'}$ element-wise as

$$c_i^{s'} = c_i^{q'}\ \ c_i^{r'} \tag{31}$$

to find the coefficients $c_i^{s'}$ describing $f_{\underline{s}}(s|D_Q;D_R)$.

- Merge the distributions $f_{\underline{s}}(s|D_S)$ (described by c_i^s) and $f_{\underline{s}}(s|D_Q; D_R)$ (described by $c_i^{s^0}$) using (27). The resulting coefficients describe $f_{\underline{s}}(s|D_Q; D_R; D_S)$ as intended.

This allows us to even take into account groups of correlated skills. In practice this is hardly ever necessary: correlated duos do occur, but triplets or quadruplets are extremely rare and often indicate an inconveniently chosen course set-up.

D. Overview of performance distribution tracing steps

We have so far seen how, given data on a skill S, its subskills $A; B; :::$ and its correlated skills $Q; R; :::$, the posterior distribution of $\underline{s}$ is determined. Let's create an overview. The entire procedure is summarized in Figure 3.

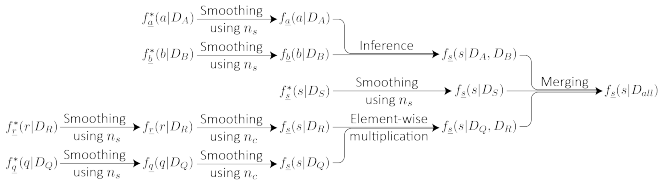

Fig. 3. Overview of how to incorporate data from subskills and correlated skills. The star distributions – stored in the database – denote the distributions *after* incorporating a new observation from an exercise, but *before* applying any smoothing for practice and time decay. For smoothing use (13), for inference use (24) and for merging use (27).

Effectively, the coefficients stored in the database are always the ones describing a distribution of $\underline{s}$ *after* data on an exercise execution has been incorporated, but *before* smoothing has been performed to take into account the learning effect (practice decay) as well as time decay. So before anything is done, the stored coefficients $\mathbf{c}^*$ are first always smoothed using (13) with smoothing order n_s. Note that, as explained in Appendix B, the value of n_s depends (among others) on the time passed since the last exercise, so it is likely to have a different value for every skill.

If a skill is a composite skill, and hence has subskills, an inference step is applied using the skill set-up. This is done through (24). Afterwards distributions are merged using (27).

Similarly, if a skill is correlated with other skills, then these other skills are first smoothed with order n_c using (13). Here n_c depends on the strength of the correlation. If there is more than one correlated skill, the resulting coefficients for each of these correlated skills are multiplied element-wise using (31). This distribution is then once more merged in using (27). The merging order is irrelevant.

Through this method, we can always find the most informed distribution of $\underline{s}$, incorporating as much data as is possible/sensible. And, because we always use distributions, the certainty/accuracy of said data is always correctly taken into account. In theory, if correlated skills $Q; R; :::$ also have subskills, or if subskills $A; B; :::$ also have correlated skills or further subskills, these can be taken into account as well. In practice, due to the repetitive smoothing, the tiny bit of extra information resulting from this is generally not considered worth the effort. Hence the plan of Figure 3 suffices.

Once the PDT algorithm has determined, for every relevant skill S, the most well-informed performance distribution $f_{\underline{s}}(s|D_{all})$, this information can then be applied. It can be used to inform a student of his/her progress, to recommend skills to practice, or to select exercises to try out. And once a new data point becomes available – the student tries and succeeds at or fails a certain exercise – update law (20) can be used to update the stored coefficients.

Note that, when using update law (20), the coefficients c_i are the coefficients of $f_{\underline{s}}(s|D_S)$ *without* any additional information, while the polynomial coefficients k'_j do have *all* available data taken into account, also from subskills and correlated skills. This may seem strange: why store the distributions $f_{\underline{s}}(s|D_S)$ and not the distributions $f_{\underline{s}}(s|D_{all})$? The reason here is practical: if we did, then when a user practices a subskill A, we would have to update *all* its parent skills. For very fundamental subskills like solving a linear equation, there may be countless parent skills, possibly also including future skills that have not been added to the system yet. As a result, storing coefficients for distributions of $\underline{s}$ based on *only* observations directly related to skill S is practically much more sensible.

V. Practical Application

To test the PDT algorithm, a web app called Step-Wise has been built that applies it. We will study how this app works, what students thought of the skill level estimates, and how the recommendations made by the app turned out.

A. General functioning of the Step-Wise app

The Step-Wise app is a practice support app. A university-level Thermodynamics course has been split up into a large learning tree of 35 skills, many with complex interdependencies. The students were taught about the theory in regular lectures. They were then instructed to practice using the Step-Wise app. For each of the 35 skills, a set of randomly generated exercises has been added, effectively resulting in an infinite amount of practice material for the students. An example Step-Wise exercise is shown in Figure 4.

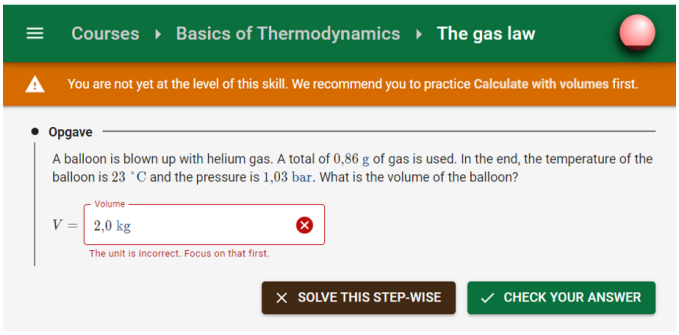

Fig. 4. A screenshot of an exercise from the Step-Wise app. Numbers are randomly generated. Every exercise is connected to a skill, and the skill level is continuously shown as a colored sphere on the top right. By hovering over the sphere, the student sees a brief explanation of what this score is and is based on. Exercises are also programmed to provide individualized feedback.

When using the Step-Wise app, students are continuously kept up-to-date on their estimated performance for every skill. This is done through the *skill indicator*: a small colored sphere,

as shown in Figure 4. For a skill S, the color of the sphere would be determined by the expected success rate $E[\underline{s}]$: red for 20%, gradually shifting to green at 80% and even turning blue on higher percentages. In addition, if the estimate was rather uncertain – the distribution of $\underline{s}$ would be flat – this color would be partly grayed out. The more certain the PDT algorithm gets of its estimates, the brighter the colors become.

Each exercise in the Step-Wise app also has an extra feature. If the student cannot figure it out, he/she can choose the option 'Solve this Step-Wise'. This splits the exercise up into steps, each representing a subskill needed to solve the exercise. The student can then do these steps one by one. Students appreciate this approach, as it helps them to better understand the content matter. Also, because they have earlier proven mastery for each of these subskills, they are certain they will always eventually be able to solve the exercise. For the PDT algorithm this Step-Wise approach proves very useful too, as it allows it to pinpoint in which subskill the student is deficient. The Step-Wise app can then recommend the student to practice this subskill some more separately.

A numerical analysis of the results of applying the Step-Wise app could not be provided. After all, the sample group (80 students) was too small, the regular variation between students was too large, and even if there were significant changes then, due to the corona crisis, these could not necessarily be attributed to this intervention. However, through a study of how students used the app, and detailed interviews held with said students, conclusions can still be drawn.

B. Experience with skill level estimates

Students are generally happy with the continuous skill level estimates provided by the app. Observations can be grouped into three categories.

- In advance, when starting to practice a new skill, the PDT algorithm can use data on subskills to estimate how proficient a student would be on that skill. One student for instance said, 'I know when I'm going to study a difficult skill, so I can mentally prepare. That helped.'
- During practice, the skill level indicators mainly motivated students to work harder. 'I really wanted the indicator to be bright green. I worked deep into the night to get there.' This was often not without any frustration. 'On a silly calculation error the app already counts it as a fail, dropping my carefully accumulated score. On an exam I do get partial points for a correct method!' This actually shows that the PDT algorithm works as intended. After all, it only counts successes and failures and not partial successes. A positive side-effect is that the students started checking their own work for errors more often, resulting in fewer calculation errors down the line.
- After practice, students were once more happy with the skill level indicator. 'It is very reassuring to have an app tell me that I have practiced enough.'

All in all, providing students with skill level estimates has shown to increase motivation, increase the time spent practicing and help students become more consistent in their work.

In general there were no cases where the PDT algorithm gave illogical skill level estimates. As a result, it can be concluded that the PDT works as intended. In addition, the data on the certainty of the estimates, as provided to the students in the coloring scheme, had a mild increase on the motivation of the students. They wanted to get the indicator to be bright green. We can conclude that this extra information was also useful, albeit only mildly.

C. Recommendations of skills and exercises

Based on the skill level estimates, students are also given recommendations on what skills to practice. If students practice a skill they have already mastered, they would be advised, 'Try this later skill instead.' Or similarly, if students would practice a skill for which a prerequisite was not mastered, they would be advised, 'You may want to try practicing this earlier skill first.' (See Figure 4 for an example of the latter.) In the thermodynamics course, the students initially liked these recommendations since it structured their learning process, and they eagerly and obediently followed them. This did reduce significantly over time. Often at some point a student would skip a skill. 'I think I already mastered it, but the app doesn't believe it, because I keep making small calculation errors.' Afterwards, the app would continuously send the student back to this skipped skill, much to the chagrin of said student. This caused the recommendations to be ignored more often. Small tweaks in the user-friendliness of the recommendation system are hence still necessary.

The Step-Wise app also includes an intelligent exercise selection method, roughly similar to [19]. Effectively, every skill has three or four exercises connected to it, each with randomly chosen parameters. Though they are similar, each exercise does have minor variations, like a different step somewhere along the way or simply an entire extra step. As a result, when we use the inference step of the PDT algorithm to estimate the success rate of all these exercises, we do get different values. The Step-Wise app was programmed to take this into account. Exercises were chosen randomly, but if an exercise had an estimated success rate close to 50% then it would be much more likely to be chosen than an exercise with a 20% (too hard) or 80% (too easy) success rate. For the teachers this add-on was positive: they noticed the students starting with the easier exercises and later on getting harder ones. However, for the students this functionality caused a more monotonous practice session. 'I keep getting the balloon exercise. Give me something else for a change!' By adding more exercises down the line, this problem can be mostly circumvented.

VI. Conclusions and Recommendations

This paper has presented the Performance Distribution Tracing algorithm. For any skill S, this algorithm continuously traces the distribution of the success rate $\underline{s}$ of this skill. This is done while taking into account the learning effect and practice decay. If there is an exercise X requiring the usage of multiple skills, in any possible set-up, then the success rate $\underline{x}$ of this exercise can also be inferred, and observations from

this exercise can be taken into account for all related skills. In addition, if domain knowledge is present, in terms of links between skills, then this knowledge can be taken into account, resulting in more informed estimates of the success rates of all respective skills.

An application to a university Thermodynamics course with 80 students has shown that the PDT algorithm works in practice. The data it provides, including data on the uncertainty of its estimates, has proven to be helpful for coaching students in their learning process, and it increases the motivation of students to practice longer and more thoroughly.

An extension of the PDT algorithm may concern individualization: can the algorithm be tuned to a specific student, resulting in better updates to give more accurate estimates? Possibly the smoothing order n_s or even the prior $f_{\underline{s}}(s)$ can be adjusted on-the-go, although the latter would require a renewed derivation of all equations in the PDT algorithm.

Another possible extension includes taking into account the cooperation of multiple students. If two students work together, and the distributions of their individual success rates $\underline{s}_1$ and $\underline{s}_2$ for some skill S are known, how can we predict the success rate $\underline{s}_{1,2}$ of the two students working together? And if they together successfully execute an exercise, how can we update their individual skill distributions? Given that the Step-Wise app actually is getting a cooperation mode, after consistent requests from students, this is a highly relevant topic for further investigation.

APPENDIX A
MATHEMATICAL DERIVATION OF COEFFICIENT EQUATIONS

This appendix has as goal to show how an expression with PDFs like (12) can be turned into an expression for coefficients like (13). We will go through the full derivation.

A. Necessary mathematics

Before we start, we must note the general relation

$$\int_0^1 g_{i;n}(x)\,dx = \int_0^1 (n+1)\binom{n}{i} x^i(1-x)^{n-i}\,dx = 1; \quad (32)$$

for $0 \leq i \leq n$. To see why this holds, first note that for $i = n$ the above relation is trivial. Next, consider any i with $0 \leq i \leq n-1$. Through integration by parts we find

$$\int_0^1 g_{i;n}(x)\,dx = \left[\frac{n+1}{i+1}\binom{n}{i} x^{i+1}(1-x)^{n-i}\right]_0^1 + \int_0^1 (n+1)\frac{n-i}{i+1}\binom{n}{i} x^{i+1}(1-x)^{n-i-1}\,dx. \quad (33)$$

The part between square brackets reduces to zero. Additionally, $\frac{n-i}{i+1}\binom{n}{i}$ simply equals $\binom{n}{i+1}$. This shows that

$$\int_0^1 g_{i;n}(x)\,dx = \int_0^1 g_{i+1;n}(x)\,dx. \quad (34)$$

By induction, relation (32) hence must also be true for all i with $0 \leq i \leq n-1$, thus proving (32) for all i with $0 \leq i \leq n$.

B. Isolating the coefficients

We now have the right basis to prove (13). To do so we must first isolate the coefficients. Our starting point is (12), for completeness repeated as

$$f_{\underline{a}_{k+1}}(a_{k+1}|D_k) = \int_0^1 \frac{f_{\underline{a}_k;\underline{a}_{k+1}}(a_k;a_{k+1})\,f_{\underline{a}_k}(a_k|D_k)}{f_{\underline{a}_k}(a_k)}\,da_k. \quad (35)$$

In this expression we have the joint prior $f_{\underline{a}_k;\underline{a}_{k+1}}(a_k;a_{k+1})$ defined by (10), or written in its sum notation as

$$f_{\underline{a}_k;\underline{a}_{k+1}}(a_k;a_{k+1}) = \frac{1}{n_s+1}\sum_{i=0}^{n_s} g_{i;n_s}(a_k)\,g_{i;n_s}(a_{k+1}). \quad (36)$$

Similarly, the posterior distribution of $\underline{a}_k$, described by $f_{\underline{a}_k}(a_k|D_k)$, follows from (3) as

$$f_{\underline{a}_k}(a_k|D_k) = \sum_{i=0}^{n} c^*_{k;i}\,g_{i;n}(a_k); \quad (37)$$

where $c^*_{k;i}$ are the coefficients describing $\underline{a}_k$. These coefficients are known from (9) or alternatively (20). Finally, we have the prior $f_{\underline{a}_k}(a_k)$ which equals 1 and can hence be ignored.

Inserting all the above expressions into (35) turns it into

$$\int_0^1 \frac{1}{n_s+1}\left(\sum_{i=0}^{n_s} g_{i;n_s}(a_k)\,g_{i;n_s}(a_{k+1})\right)\left(\sum_{i=0}^{n} c^*_{k;i}\,g_{i;n}(a_k)\right)da_k. \quad (38)$$

We can rearrange the above to write it as

$$\sum_{i=0}^{n_s}\left(\frac{1}{n_s+1}\int_0^1\left(\sum_{j=0}^{n} c^*_{k;j}\,g_{j;n}(a_k)\right)g_{i;n_s}(a_k)\,da_k\right)g_{i;n_s}(a_{k+1}). \quad (39)$$

Comparing this with the standard basis function notation (3), we can directly see that the coefficients $c_{k+1;i}$ describing $f_{\underline{a}_{k+1}}(a_{k+1}|D_k)$ equal

$$c_{k+1;i} = \frac{1}{n_s+1}\int_0^1 \sum_{j=0}^{n} c^*_{k;j}\,g_{j;n}(a_k)\,g_{i;n_s}(a_k)\,da_k. \quad (40)$$

The coefficients have hence been isolated. What remains is solving the integral.

C. Solving the integral

Consider (40). Note that, if we normalize the coefficients $c_{k+1;i}$ afterwards, we can safely ignore constant multiplications: factors not depending on i. Hence $\frac{1}{n_s+1}$ drops out. If we expand the basis functions using their definition (2), and ignore the constant factors $(n+1)$ within, we can write the above expression for $c_{k+1;i}$ as

$$\int_0^1 \sum_{j=0}^{n} c^*_{k;j}\binom{n_*}{j} a_k^j(1-a_k)^{n-j}\binom{n_s}{i} a_k^i(1-a_k)^{n_s-i}\,da_k. \quad (41)$$

Rearranging the above turns it into

$$\sum_{j=0}^{n} c^*_{k;j}\binom{n_s}{i}\binom{n_*}{j}\int_0^1 a_k^{i+j}(1-a_k)^{n+n_s-i-j}\,da_k. \quad (42)$$

The integral can now directly be solved through (32). If we once more ignore constant multiplications, we find

$$c_{k+1;i} = \sum_{j=0}^{n} \frac{\binom{n_s}{i}\binom{n}{j}}{\binom{n+n_s}{i+j}} c^*_{k;j}. \tag{43}$$

By applying the definition of the binomial, the above can be expanded into

$$c_{k+1;i} = \sum_{j=0}^{n} \frac{\frac{n_s!}{i!(n_s-i)!}\frac{n!}{j!(n-j)!}}{\frac{(n+n_s)!}{(i+j)!(n+n_s-i-j)!}} c^*_{k;j}. \tag{44}$$

Rearranging terms turns this into

$$c_{k+1;i} = \sum_{j=0}^{n} \frac{\frac{(i+j)!}{i!j!}\frac{(n+n_s-i-j)!}{(n+n_s)!(i+j)!}}{\frac{(n+n_s)!}{n!n_s!}} c^*_{k;j}. \tag{45}$$

The denominator here equals $\binom{n+n_s}{n}$, which is a constant, so it can safely be ignored. We remain with

$$c_{k+1;i} = \sum_{j=0}^{n} \binom{i+j}{i}\binom{n_*+n_s-i-j}{n_*-j} c^*_{k;j}, \tag{46}$$

which is the final result that we wanted to get. Note that in theory it is possible to take into account all constants in the derivation of $c_{k+1;i}$ too, but in practice it is much easier and more sensible to ignore constants and simply normalize coefficients afterwards.

APPENDIX B
CHOOSING AND APPLYING THE SMOOTHING ORDER n_s

In Section II-C it was discussed that, to incorporate the learning effect and practice decay, the distribution of the success rate $\underline{a}$ would be smoothed after every exercise. This is done with some smoothing order n_s. The question remains: how should n_s be chosen? This is not a mathematical matter, but one of settings and preferences.

A. Links between original and smoothed distributions

Consider a skill A. Let's write the success rate *before* smoothing as $\underline{a}_k$ and the success rate *after* smoothing as $\underline{a}_{k+1}$. In this case, based on (13), a link can be determined between their expected values $\mathrm{E}[\underline{a}_k]$ and $\mathrm{E}[\underline{a}_{k+1}]$. To be precise, this link satisfies

$$\mathrm{E}[\underline{a}_{k+1}] - \frac{1}{2} = \frac{n_s}{n_s+2}\left(\mathrm{E}[\underline{a}_k] - \frac{1}{2}\right). \tag{47}$$

In words, the smoothed success rate $\underline{a}_{k+1}$ always has a mean closer to $1/2$ than the original success rate $\underline{a}_k$. To be precise, the reduction factor that is applied is

$$r = \frac{n_s}{n_s+2}. \tag{48}$$

We call r the *decay ratio*. If it equals 1 there is no smoothing, while a decay ratio of 0 means everything is forgotten. There are now two questions: which decay ratio r is appropriate? And how do we turn it into a smoothing order n_s?

B. Choosing a decay ratio

Smoothing should take into account both the learning effect and time decay. For time decay an exponential decay seems appropriate. Something like

$$r = (1/2)^{t/t_{1/2}}; \tag{49}$$

where t is the time since the last exercise and $t_{1/2}$ is the time after which the student has 'lost' half of its skills. The latter is generally set to a year.

For practice decay, we could introduce another factor. But instead, slightly more intuitively, we use the concept of *equivalent time*. We say that practicing one more exercise is equivalent to a time t_e (set as two months) of not practicing. We call t_e the *equivalent inactive time*. This gives a decay ratio of

$$r = (1/2)^{(t+t_e)/t_{1/2}}. \tag{50}$$

However, we still go one step further. The first time a student does an exercise, the learning effect is still strong. It's the student's first time, so he/she is expected to learn a lot from it. However, after ten exercises or so, this effect is a lot smaller, and exercise fifty is unlikely to still have much of a learning effect at all. To incorporate that, we reduce the equivalent inactive time t_e based on the amount of practice a student has had. If the student has already practiced the skill n times, then we define

$$t_e = t_{e;0}\,(1/2)^{n/n_{1/2}}. \tag{51}$$

Here $t_{e;0}$ is the *initial equivalent inactive time*, set to two months, and $n_{1/2}$ is the number of times a student must practice a skill to get half the learning effect. Generally we set $n_{1/2} = 8$, although ideally this is larger for smaller skills and smaller for larger skills.

Using the above equations, we can always calculate an appropriate decay ratio r. It only needs to be applied.

C. Applying a decay ratio

To apply a given decay ratio r, we could simply choose a smoothing order. From (48) you would expect it to equal

$$n_s = \frac{2r}{1-r}. \tag{52}$$

However, the smoothing order n_s must be an integer, which complicates matters. A rough solution would be to round n_s to the nearest integer value and apply that, but this results in discontinuities. Perhaps, from one day to the next, a student's score suddenly jumps. That is not desirable and may result in confusion/frustration.

A cleaner and more continuous solution would be to write $r = r_1 \oplus r_2 \oplus r_3 \oplus \cdots$, for which each individual decay subratio r_i does result in an integer smoothing order $n_{s;i} = \frac{2r_i}{1-r_i}$. We can then apply smoothing multiple times. Choosing the subratios $r_1, r_2, \ldots$ is done by applying the following steps for $i = 1, 2, \ldots$.

- Choose the smoothing order $n_{s;i} = \lceil \frac{2r}{1-r} \rceil$.
- Pick $r_i = n_{s;i}/(n_{s;i}+2)$ accordingly.
- Update the remaining decay ratio r using $r \ominus r_i$.
- If $n_{s;i} > n_{s;max}$ then stop: ignore r_i and further.