Learning a Skill-Teaching Curriculum with Dynamic Bayes Nets

Derek T. Green and **Thomas J. Walsh** and **Paul R. Cohen**
Department of Computer Science, University of Arizona, Tucson, AZ 85721

Yu-Han Chang
Information Sciences Institute, University of Southern California, Marina del Rey, CA 90292

Abstract

We propose an intelligent tutoring system that constructs a curriculum of hints and problems in order to teach a student skills with a rich dependency structure. We provide a template for building a multi-layered Dynamic Bayes Net to model this problem and describe how to learn the parameters of the model from data. Planning with the DBN then produces a teaching policy for the given domain. We test this end-to-end curriculum design system in two human-subject studies in the areas of finite field arithmetic and artificial language and show this method performs on par with hand-tuned expert policies.

Introduction

This paper considers an Intelligent Tutoring System (ITS) for teaching a student to master multiple skills by presenting hints and problems to the student. Unlike previous ITS systems that focused on difficult multi-step problems (Conati, Gertner, and Vanlehn 2002), once the skills in our setting are acquired, the problems themselves are fairly easy to answer. The difficulty is for the system to determine what problems to ask or what hints to give based on the student's *proficiency state* with a goal of helping the student master all of the skills quickly. We call this problem *curriculum design*.

In practice, when teachers design curricula they typically divide subjects into lessons and order the presentation of the lessons to best aid in learning. However, human teachers are faced with the challenge of ordering their lessons to help students as a group and are often unable to spare time for individual student needs. This poses a dilemma since individual one-on-one teaching is considered far more effective than typical classroom style teaching (Bloom 1984).

In our approach to this problem, we mine statistics from previous student training data in order to model the dynamics of skill acquisition. From this data, we construct a Dynamic Bayes Net (DBN) (Dean and Kanazawa 1989), a popular representation from the reinforcement learning community that has been used in previous work on *tracking* student skill acquisition (Almond 2007). This factored model allows us to generalize our mined knowledge and to consider curriculum design as a sequential decision making problem, where an optimized policy maps states of student knowledge to the most effective lesson (problem or hint) to pose next. In effect, the policy acts on two levels, it encodes the overall lesson organization but since it is state-based it can adjust to individual students as their proficiencies change.

Such a curriculum design system is appropriate for individual use either online or in a classroom setting. It can be used to complement regular classroom work, or as a standalone tutorial system. The structure we describe will be effective for any subject matter which can be broken down by a domain expert into subtopics between which there are inherent ordering dependencies (i.e. any learning scenario in which learning builds upon itself.) In the conclusion section we discuss how to adapt this work to new domains.

To demonstrate the effectiveness of our approach, we ran human-subject trials in two skill-acquisition domains where students were unlikely to have strong prior knowledge. Our results showed the DBN-based approach performed as well as several expert policies and significantly better than the random baseline. Also, the generality of our approach can be seen in that only small modifications from our skill-teaching DBN template need to be made to tailor our model to these specific domains. The major contributions of this work are a generalized DBN representation and algorithms for curriculum design, as well as human-subject trials verifying the validity of this approach.

Problem Definition and System Overview

The curriculum design problem involves a teacher interacting with a student in a domain that contains a set of *skills* $\mathcal{S}$. Each skill s has a set of dependencies $D(s)$, the members of which are other "pre-requisite" skills. For instance, in an arithmetic domain, the skill of multiplying might have addition as a pre-requisite. In this work, we assume that D maps each skill only to the skills that it directly depends on, and that the dependency graph is acyclic. There is also a set of *facts* F, which may be necessary for answering problems or interpreting hints about the skills. For instance, in a language domain where the skills involve constructing sentences from a given set of words to describe a picture ("The circle is blue"), the student's vocabulary would be considered a set of facts (knowing the meaning of blue), while the ability to conjugate and place the words in the sentence would depend on a number of skills.

The actions available to a teacher are to either provide a *hint* $H(s)$ or ask a *problem* $P(s)$ about skill s and see the student's answer. Note the teacher will not necessarily be able to choose the exact hint or problem given to the student, rather a particular grounded problem will be randomly selected from all available problems about s. This is because our goal is for the teacher to lay out a general curriculum or *policy* (e.g. when to ask a multiplication problem) and because the number of grounded problems may be infinite.

From a history of such interactions (or *trajectory*), the teacher can model a student's *proficiency state* ρ, composed of the proficiency on each skill $\rho(s)$. In general our approach to such tracking will involve a set of *proficiency adjustment rules*. These rules specify how the teacher updates $\rho(s)$ based on the history of the teacher-student interaction. For instance, if the teacher stores proficiency values as integers, answering a problem from $P(s)$ correctly might increment $\rho(s)$ by 1, and a wrong answer could decrement it. The goal of a teacher will be to, with as few problems and hints as possible, have the student achieve the highest proficiency value possible on each skill ρ_{max}.

Our overall system uses the dependencies $D(s)$ in conjunction with a set of trajectory data from students trained using hand-crafted problem presentation policies. A multi-layered DBN that models the dynamics of ρ is constructed using D to define the structure. The student data is used to build the conditional probability tables (CPTs). The learned DBN model is used by a planning algorithm which determines a teaching policy, mapping ρ to actions. This dynamic curriculum is then used to train students who are evaluated on a series of tests covering a wide range of skills.

A Multi-Layered DBN Representation

Our representation models a student's changing proficiency levels using a multi-layered Dynamic Bayes Net (DBN). DBNs are used in reinforcement learning to represent transition functions for state spaces comprising multiple factors. A traditional Markov Decision Process (Puterman 1994) $M = \langle X, A, R, T, \gamma \rangle$ models an environment with states X, actions A, reward function $R : X \mapsto \Re$, transition function $T : X, A \mapsto Pr[X]$ and discount factor γ. However, if the states comprise n factors $x_1 ... x_n$, then a tabular representation of T will be exponentially large in n. DBNs dodge this curse of dimensionality by assuming that each factor is conditioned only on a small number ($k = O(1)$) of other *parent* factors $\phi(x_i)$. This results in a distinctive 2-layer architecture with next state factors x'_i linked to parents in the current timestep $\phi(x_i)$ and the probabilities of individual factor transitions stored in CPTs. The full transition function for an MDP is recoverable from the DBN as: $T(x, a, x') = \prod_{i=1}^{n} Pr[x'_i | \phi(x'_i), a]$. Notice the number of parameters that need to be learned to specify the DBN representation is only $O(n^k)$, far smaller than the fully enumerated state space.

Unfortunately, a 2-layer DBN will not capture several important aspects of the domain, including some correlated effects. We will resolve these and several other issues by introducing a more complex multi-level DBN structure.

Skill Teaching DBNs

Modeling skill acquisition as a multi-level DBN has been proposed in prior work (Almond 2007) but the DBN was constructed entirely (structure and CPTs) by hand and used for state tracking with a hand-coded policy and an artificial student. By contrast, we will show how the CPTs of a differently structured DBN can be *learned* from data and *planned* with, to create a policy that we then test on human subjects. We begin by mapping the parameters of the DBN to the components of the curriculum design problem.

The state of our factored MDP consists of one factor for each skill $s \in \mathcal{S}$ corresponding to the student's proficiency $\rho(s)$ on that particular skill. By having a state factor for the proficiency on each skill (such as the ability to multiply 2-digit numbers), the system will be able to dynamically choose problems or hints based on the student's current capabilities. The nodes corresponding to these factors can be seen in the top and bottom rows of Figure 1, corresponding to the state of the agent at time t and $t + 1$, respectively. Skill dependencies are encoded by edges to layer $\rho_{t+1}(s)$, either directly (node T_0 in Figure 1) or indirectly (through the Σ-Dep nodes in Figure 2). This dependency structure means that the value of a skill on the next timestep will depend on the proficiency values of all of these related skills. We note that these edges only encode potential dependencies between skills and likely do not on their own determine the optimal policy. The actions in the DBN (the diamond shaped node) are all the $H(s)$ or $P(s)$ (sets of hints or problems about a skill) that can be chosen by the teacher. Rewards in our skill-teaching DBNs are simply $R(\rho_t) = \sum_{s \in \mathcal{S}} w(s) \mathbb{I}(\rho_t(s) = \rho_{max})$, that is a weighted sum of the number of skills that have been mastered so far by the student. In our studies we always used $w(s) = 1$.

Intermediate Nodes

We will use intermediate nodes to simplify and correlate the calculation of next state probabilities given a current proficiency state and an action. These nodes will generally reduce the number of parameters in the DBN and will help capture correlations and interactions between parts of the model.

Matching, Aggregation and Correlation We first consider intermediate nodes for matching skills to the problems or hints specifically about those skills. In a typical propositional DBN, factors are usually linked to the action node itself, but this would add $2|\mathcal{S}|$ possible parent values to each CPT and most of these entries would be redundant, because if the action is not $H(s)$ or $P(s)$ there is often no effect on $\rho(s)$. Hence, we introduce a *Match* node for each skill (yellow nodes in the figures), that indicates whether the action is a hint for s, a problem on s, or not specifically about s. In this way, the CPT for $\rho(s)$ needs only to be increased by a factor of 3 rather than $2|\mathcal{S}|$.

Another important intermediate node type for limiting CPT size is the *Aggregation* node, which counts the number of parent nodes with some value. For instance, the green Σ-Dep nodes in Figure 2 count the number of parent skills for a $\rho(s)$ that have not reached the mastery level. This allows nodes that may depend only on the number of mastered pre-requisite skills to consider just this count, rather than all possible combinations, leading to an exponential decrease in the size of the corresponding CPTs.

Another modeling challenge is that if the student answers a problem, multiple skill proficiencies might need to be updated, so we must ensure that these changes will be synced. These correlations are enforced by having all the skills that might be affected by such a change linked directly (in the same layer) to the next proficiency factor for the skill s. Such correlations can be seen in the bottom layer of Figure 2.

Intermediate Nodes Based on Ground Facts While our goal in this work is to build policies for student skill acquisition, the ground problems we present also have facts in them. For instance, answering problems in our artificial-language domain requires a student to know the meaning of the words, not just how to order them. We do not want to lower the proficiency rating on a skill if the facts used in the ground action were not known, since an unknown fact is likely to blame. Thus we introduce *FactsKnown* nodes for each type of action (shown as red nodes in Figure 2), which, when a ground action is presented, take on a binary value based on whether all the facts in the problem are known or not. These nodes are then attached to the corresponding skill proficiencies to handle the credit assignment described above and to provide finer granularity in the CPT for advancing a skill (some skills may be easier to learn than others when few facts are known). When keeping track of fact proficiencies is prohibitive, or during the planning phase when answers to specific questions will not be tracked, one can estimate the probability of knowing a fact based on the number of skills known, a technique we utilized in the planning component of our second case study.

In summary, we introduced a number of important intermediate nodes and structures into the Skill Teaching DBN template that are generally needed in modeling skill acquisition. These included Match nodes to couple actions and skills, Aggregation nodes for generalizing on dependency proficiencies, FactsKnown nodes, and correlation links in the ρ_{t+1} layer. Generally, these structures keep the sizes of the CPTs small, model correlations in action outcomes, and keep track of the interactions between facts and skills.

Curriculum Design as Planning

Unlike other approaches that track a student's state and map this to a hand-coded policy, we will be using our learned DBN model to create the teaching policy itself. We will do so by treating curriculum design as a planning problem with reward and transition functions specified with a DBN as described above. Viewed in this manner, we can calculate the value of each action a from a given proficiency state ρ as $Q(\rho, a) = R(\rho) + \gamma \sum T(\rho, a, \rho') max_{a'} Q(\rho', a')$, and our policy is derived by choosing the maximum valued a at a given ρ. Intuitively, these values tell us what action will help us achieve full proficiency fastest, while also (through the discount factor) coveting skill proficiencies that can be learned sooner. This implicit trade-off between long-term teaching goals and short term skill proficiency is a key component of our approach and recognizes the practical limitation of short teaching sessions. In our experiments we used Value Iteration (Puterman 1994) to compute the Q-values and the corresponding policy, but this approach was at the limit of tractability (several hours), and future iterations with larger state spaces will require approximate planners specifically designed for DBNs, e.g. (Hoey et al. 1999).

Learning DBN Parameters from Trajectories

While the DBN architecture will be specified using the node types above, the parameters for the individual CPTs will be mined from data. In our studies, we will train the CPTs on data from a series of "expert" and random policies as well as some policies where users choose the problems or hints. The mix of these data streams is important because mining data collected from a single specific deterministic policy would lead to simple mimicry of that policy due to "holes" in the CPTs where we have no evidence as to whether or not a problem will be answered correctly in a given context. The use of human-guided exploration in the "choice" policies (where students could select from a set of problems) also yields information in many new areas that might lead to better policies. In states where such "holes" still exist, we will fill in the holes as "no change in state" to keep the agent in known areas of the state space. This discouragement of autonomous exploration is necessary because we need to run human trials to collect data and many (overly optimistic) exploratory policies (Kearns and Koller 1999) are likely to be unhelpful or even detrimental. Exploration issues are discussed further in the conclusion.

Case Study I: Finite Field Arithmetic

We begin with a case study on finite field arithmetic (FFA)[1], a simple domain where we made a number of assumptions about the dynamics (hard coding many rules in the CPTs), most of which are relaxed in the second case study. FFA problems consist of mathematical operators $(+, *, -, /)$ applied to a finite alphabet (in our case $\{0, 1, A, B\}$). While some problems are intuitive to answer $(A + 0 = A)$, others require learning $(A * B = ?)$. To reduce the size of the domain and to ensure a clear skill hierarchy, we limited the size of problems to 2 operators and we constructed a series of *chains* of related problems. A chain consists of a primary problem (e.g. $A * B = 1$), a secondary problem involving the primary problem ($(A * B) - A$), and a tertiary problem containing the secondary problem with its primary part reduced (e.g. $A/(1 - A)$ from substituting $A * B = 1$ into $(A * B) - A$). The action set contains problems (with the answer revealed afterwards), and there are no hints. In this way, we enforce a prerequisite hierarchy among the skills (e.g learning anything from $(A * B) - A$ is quite difficult without already knowing that $A * B = 1$).

Viewed in this manner, the domain has no facts, and the skills are all the primary, secondary and tertiary problems

[1]Specifically, a "Galois Field" of size 4 or $GF(2^2)$, although an understanding of Finite Fields is not necessary for the readers of this paper, nor for the subjects in this study

from each chain and some "trivial" problems (like $A + 0$). The proficiency value of each skill is 0 or 1 and we discuss the adjustment rules below. The dependencies D link the problems within each chain as illustrated in the DBN in Figure 1. Because of the small parent structure and absence of facts, the only internal structure is from *Match* nodes.

We collected training data for the DBN using a number of hand-coded policies, specifically a random policy, an expert policy represented as a finite state machine (FSM) that grouped primary and secondary problems from the same chain, and two "fixed orderings" without backtracking. One of the fixed orderings kept problems in the same chain together and performed as well as the FSM, while the other mixed problems from different chains and was on par with random. During a trial, the proficiency adjustment rules increased $\rho(s)$ from 0 to 1 after 3 consecutive correct answers and decremented it for 2 consecutive wrong. However, to get as much out of the data as possible, in our training we built CPTs where 1 right or wrong answer produced the transition, a technique that was extremely helpful in larger state spaces (as in the next case study). We also assumed that secondary or tertiary problems could not be answered correctly without their dependencies being mastered (this deterministic effect of dependencies is relaxed in the second case study). Thus, the main problem left to the DBN in this simplified domain was choosing what chain to pick the next problem from.

Interestingly, unlike the FSM policy (the best of the hand-coded techniques), the learned DBN policy "swapped" between chains, but its policy was far more successful than the fixed ordering that also tried to swap. The DBN taught most of the primary problems first (in order of the easiest to learn based on the data), then the secondary and the more difficult tertiary problems. This tendency reflects the DBN's preference to teach easy skills first (an effect of the discounted value function), a trait that is crucial in domains with a finite amount of training time.

We collected results for our DBN policy and compared them to the hand-tuned policies using a 15 point pre-test (before any training) and post-test set of questions where, unlike in training, students were not told the correct answers. We used a subject pool of university undergraduates, mostly between ages 18-20, all taking an introductory psychology course and receiving modest class credit for participation. The subjects were trained on 24 problems as chosen by the teaching policy between the pre and post tests. The means and standard deviations for the improvement (post-test minus pre-test) with the Random, FSM and DBN policies were $2.964(0.57)$, $4.692(0.58)$ and $5.037(0.48)$ respectively. A standard t-test shows (with $p < 0.05$) that there was a significant different in improvement between random and the other two policies.

Since we were limited to relatively small sample sizes (26 in both) we further explored the FSM and DBN data using resampling. By selecting with replacement from the FSM sample we generated a new sample of 26 students, FSM$'$. We performed the same operation on the DBN group to give us DBN$'$. We then calculated the ratio of the delta values for the two new samples as $\delta_{DBN'}/\delta_{FSM'}$ and repeated this

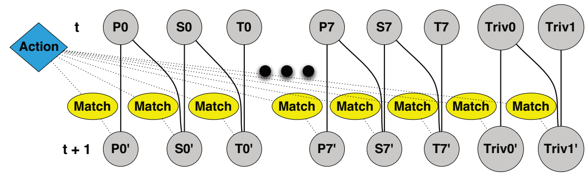


Figure 1: FFA Study DBN Structure and sample CPT

process for $100,000$ bootstrapped samples. If the means of the two groups are equal we would expect the ratio to be 1.0, but the upper and lower 2.5% quantiles yielded ratios of approximately 90% to 132% implying that the performance of the students under the DBN policy is at 90-132% of the performance of those under the FSM policy.

Case Study II: Artificial Language

Our second domain focused on the syntax and semantics of an artificial language. The language contained only a few words (few facts), but the words could be ordered to construct phrases with very different meanings (many skills).

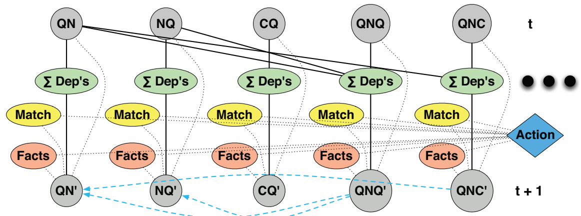
Language Description

The language incorporates three types of words: nouns, color modifiers, and quantity modifiers. Each of three nouns (N) refers to a simple geometric shape: "*bap*"= □,"*muq*"= △,"*fid*"= ○. Three color modifiers (C) refer to colors ("duq" = orange, etc.). Colors are used as postfix operators on nouns (e.g. "*muq duq*"= ▲).

Three quantity modifiers (Q), each of which is polysemous, have the following meanings: "*oy*"={small, one, light},"*op*"={large, many, very},"*ez*"={not, none, non}. The specific meaning of a Q-modifier depends on the context. As a prefix to an N (i.e. QN) a Q signifies the size of the noun, (e.g. "*op muq*"= ▲ "a large triangle"). As a suffix to an N, (i.e. NQ), it signifies the cardinality of the N, (e.g. "*muq oy*"= ▲ "one triangle"). As a suffix to a C, (i.e. CQ), it signifies the intensity or saturation of the C (e.g. "*muq duq op*"= ▲ "a very orange triangle"). Multiple Q-modifiers can be used in a single phrase as in "*op muq op nef oy*"= ▲▲▲ "many large light-green triangles".

Skills, Facts and Actions The skills in this domain are the ability to construct or understand all the kinds of phrases up to length 5. That is, $\mathcal{S}$ is the set $\{N, C, NC, NQ, QN, QNC, ..., QNCQ, QNQCQ\}$. The dependency structure D simply links every skill s of length l to any skill of length $l - 1$ that is a substring of s The state space is defined by the proficiency on each skill, giving us the top and bottom layers of the DBN in Figure 2.

There are 3 possible values for each $\rho(s)$, and the adjustment rules used during testing were that 3 correct answers in a row incremented $\rho(s)$, and 1 wrong answer decremented it if the facts were known. A hint increased the student's proficiency to a value of 1 (but not higher). Finally, a wrong answer for a skill that had $\rho(s) = 0$ would decrement the dependent skills. Since we are interested in learning skills (not facts), a table of definitions for all of the N and C words



t				t + 1		
Match	Facts	Σ Dep	QNQ	p(QNQ = 0)	p(QNQ = 1)	p(QNQ = 2)
0	*	*	0	1.0	0.0	0.0
*	0	*	0	1.0	0.0	0.0
H(QNQ)	1	0	0	0.0	1.0	0.0
P(QNQ)	1	0	2	0.0	0.1	0.90
P(QNQ)	1	1	2	0.0	0.6	0.4
P(QNQ)	1	2+	0	0.90	0.90	0.90

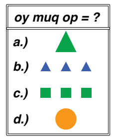


Figure 2: A partial DBN (and partial CPT for QNQ') for the artificial language domain, along with a sample problem.

was given to all students, so the only facts to be learned were the Q words.

As for the actions, hints took the form of a phrase paired with its meaning while problems came in three types, chosen randomly once a $P(s)$ was selected. In the first type (shown in Figure 2), a phrase was presented followed by a list of possible meanings. In the second, a meaning was followed by a list of phrases. In the third, a phrase with a single word replaced by a blank paired with a meaning was followed by a list of possible words to fill in the blank.

DBN Construction and Training

The state factors for the DBN are simply the proficiencies on the skills with dependencies as described above. Several of these (like N), are assumed to always be mastered, since they are given to the student.

The internal structure of the DBN (Figure 2) contains all three types of intermediate nodes defined earlier. This includes the *Match* nodes linking hints and problems to their skill, *Aggregate* nodes for counting the parent nodes below mastery level (up to 2), and *FactsKnown* nodes. This leads to a DBN with fewer parameters and stronger generalization, at the risk of some extrapolation. In our studies, we instantiated the *FactsKnown* nodes based on the same proficiency adjustments as the skills, except once mastery was reached on the facts, it was assumed they would never decrement. During the planning phase (when ground facts are not considered nor tracked), the probability of knowing the facts was set based on the number of skills mastered so far. These probabilities were based on our subject data. Finally, we want a wrong answer to $P(s)$ to be able to decrement the skill proficiencies in $D(s)$ if $\rho(s)$ is already 0. The edges linking nodes in the bottom layer of the DBN enforce these correlated changes. A partial CPT for $\rho'(QNQ)$ is shown in Figure 2.

Training As in the FFA study, the CPT parameters were learned from traces of other policies, including a random policy and several expert variants. These expert policies kept track of ρ and used the same proficiency adjustment rules as the DBN, but used a fixed skill ordering (constructed by the language designers). At each step, they selected a problem for the lowest ordered skill not yet mastered (or randomly when all were mastered), choosing a hint only when a $\rho(s)$ was decremented to 0.

We also experimented with versions of the expert and random policies that offered a choice of 3 problems/hints to a student. This was done both to better engage the students and, when the choices were over different skills, to act as a form of human-guided exploration in the training. In the first version (called "CHOICE"), the expert or random policy was used to select a grounded $P(s)$ or $H(s)$ as before, but in both cases, two other ground problems from $P(s)$ were also presented as alternative problems. In this initial presentation, the meaning in the hint or the multiple choice answers were not revealed. The student selected one of these problems and answered it as before. Using the idea of the zone of proximal development (Vygotsky 1978), we also added the "CHOICE ZPD" condition to just the expert policy, in which one of the alternative problems came from a higher-order skill in D that was not yet mastered.

When training the CPTs, we again used a different proficiency adjustment rule for incrementing $\rho(s)$, by assuming a single answer could increment or decrement $\rho(s)$. This was done because keeping track of all the stages with the "3 in a row" rules made the state space too large, and when ignoring the stages the data with the original rule was too sparse to allow deviation from the expert policies. We also expressly disallowed increments in $\rho(s)$ when 2 or more dependencies of s had not been mastered. The resulting DBN policy differed from the training policies in a number of ways. First, the DBN policy exploits the deterministic effect of hints by giving them whenever a $p(s)$ is at 0, thereby avoiding the loss of proficiency of the pre-requisite skills. Interestingly, the DBN policy also chose a different ordering for teaching the skills, specifically grouping related length 2 and 3 skills where only a C or N is added (e.g. teaching QN and then QNC next).

Evaluation and Results

For each of the policies, students drawn from a pool of university undergraduates, most participating for credit, completed four 7-minute training sessions with each session followed by a 20-question test. Unlike in training, answers to the test questions were never given to the students, though their score on each test was. All students completed the first two training/test sequences. As an incentive, beginning with the second test, any score of 90% or higher allowed the student to leave the experiment early The final test score of students that "tested out" was propagated forward replacing any missing scores for analysis.

Our analysis will focus on several factors and their interactions. By *condition* we mean a combination of two factors: the **policy** by which the teacher selects problems (either *random*, *expert*, or *DBN*) and the type of **Choice** provided to the student as described earlier. In all, we have six conditions: *Random-NoChoice* , *Random-Choice*, *Expert-NoChoice*, *Expert-Choice*, *Expert-Choice-ZPD* and *DBN-NoChoice*. Our dependent measure for most analyses is the increase in the number of correct problems between the first test and the last.

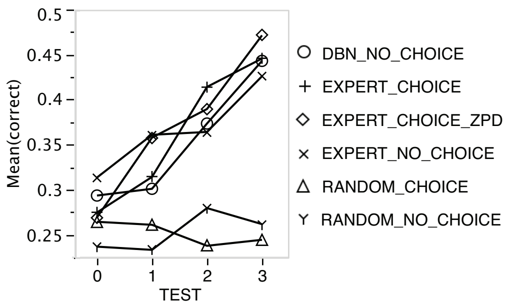


Figure 3: Mean score on each of four tests by condition.

Comparing Policies Overall, students improve between the first test and the last in the *Expert* and *DBN* conditions but not in the *Random* conditions. Mean scores on each of the four tests are plotted by condition in Figure 3. In the *Random* conditions, students' performance on the tests hovers around chance (0.25) and does not improve. This result establishes that the policy for presenting these artificial-language problems matters: if it is a random policy, students don't learn; if it is an expert or DBN policy, they do.

The amount and rate at which students learn in the *Expert* and *DBN* conditions do not seem very different. Students in all of these conditions start at roughly chance performance on the first test and are able to answer roughly 45% of the questions correctly on the final test. A two-way analysis of variance comparing *DBN* with the other *Expert* policies crossed with test number shows a main effect of test number ($p < .0001$) and no main effect of *DBN* nor any interaction effect. That is, students improve from one test to the next but whether they work under the *DBN* policy or any other expert policy makes no difference to their improvement.

So is the *DBN* policy as good as the other expert policies? This is a confidence interval question: We will show that the confidence interval around the difference between the policies is small and contains zero. Let $\chi_{s,q,t} = [0, 1]$ represent whether student s answered question q on test t correctly (1) or incorrectly (0). Let $\pi_{\bullet,q,t}$ be the mean number of correct answers, averaged over students, for question q on test t, and let $\iota_q = \pi_{\bullet,q,3} - \pi_{\bullet,q,0}$ be the mean *improvement* on question q between the first test (test 0) and the last (test 3). The average value of ι_q for *DBN* students is 15% and for the other expert policies is 16%. The confidence interval around ι_q is $[-0.06, 0.09]$. This means that, by question, the difference in improvement under the *DBN* and other expert policies ranges from -6% to 9% with 95% confidence. Another way to look at the difference between *DBN* and the other expert policies is to ask how much of the variance in improvement does the difference explain. In a two-way analysis of variance, where the factors were policy (*DBN* and *NotDBN*) and test question (with 20 levels), the mean square error for the first factor was 0.1 and for the second was 1.14. This means that the test questions account for more than ten times as much of the variance in improvement as the policy. It is harder to improve on some test questions than on others, and the policy – whether it is *DBN* or another expert policy – has a small effect relative to the differences in improvement due to the test questions themselves. In sum, the *DBN* policy's performance is indistinguishable from the other expert policies, at least with respect to improvement from the first test to the last.

Related Work

Many traditional ITS systems that follow students as they work through a specific type of multi-step problem (e.g. solving a physics problem) utilize Bayes Nets in some capacity, usually to model the partially hidden state of the student. For instance, Bayes Nets (but not MDP-style DBNs) have been used in a physics tutoring system (Conati, Gertner, and Vanlehn 2002) to keep track of the probabilities that certain facts and skills are known and use a rule-based policy to give hints based on this belief state. Others have employed a data-driven approach (like our learning system) to train such Bayes Nets from expert trajectories, for instance in an ITS system for number factorization (Manske and Conati 2005). However, these works did not treat the problem as a traditional planning problem, instead using a fixed rule-based policy with the Bayes Net tracking student state.

Other approaches have considered skill acquisition through the lens of an MDP. Work on tutoring students for completing logic proofs (Barnes and Stamper 2008) used an MDP to model the state of the student and planned out a policy for when to give hints. However, that work was again focused on a single type of multi-step problem and used a tabular representation of the MDP (not a DBN). The closest work to our own is the approach of (Almond 2007), who used a multi-level DBN (a "bowtie" Net) to model the skill acquisition process. However, instead of using simple proficiency rules as we did, that work considered skill proficiencies to be partially observable and used a particle filtering method to link the observations and underlying states. Also, in that work the DBN parameters and policy were hand-made (though planning was discussed) and the only experiment was run on an artificial student. By contrast, our DBN is learned from data, and planning is used to compute a policy that we then tested on human subjects. Approaches in other areas have used similar architectures with multi-level learned DBNs and planning. Examples include using learned DBNs to suggest actions to a welfare case worker (Dekhtyar et al. 2009) and helping dementia patients complete daily tasks (Hoey et al. 2011).

Future Work and Conclusions

This work establishes a general template for curriculum design using a DBN trained from user trajectories and for building a policy for issuing hints and problems to students based on their state of proficiency. We presented case studies in two artificial domains that contain characteristics of several real world mathematics and language environments. In the future, we plan to deploy our system in a large-scale online tutoring setting, the AnimalWatch ITS (Beal et al. 2010), which has proven to be effective in teaching algebra readiness mathematics, including basic arithmetic, fractions, variables and expressions, statistics and probability, and simple geometry (www.animalwatch.org). With roughly 3000 interactions a day, collecting data to train (and re-train) our models in this setting could be done much faster than in our current human-subject studies.

When expanding our architecture to this and other real world domains, a number of design decisions must be made, which we now briefly outline. First, the skills and facts need to be identified, a task which may be non-trivial. For instance, in a mathematics domain, while being able to multiply is a skill, determining at what level multiplication problems should be treated as atomic facts (as with elementary multiplication tables) versus a general skill, requires some expertise. Identifying the dependency structure of the domain is the next crucial step for the use of our system. The sparser the pre-requisite links are, the fewer parameters the system needs to learn, so there is a benefit to modeling as few dependencies as possible while having enough of them to make sure the DBN captures the correct conditional probabilities. Automating this construction may be aided by techniques for mining the DBN structure from the trajectories (Degris, Sigaud, and Wuillemin 2006). The particular proficiency adjustment rules used will depend on the effectiveness of predicting skill proficiency through observations of student answers to problems. Choosing the value ranges for the intermediate nodes, including the maximum sum for the Aggregate nodes, can also affect the performance of the system by making parameter space more granular, but requiring more data to train. Finally, because our system has no autonomous exploration policy during learning, its training data is biased by whatever policies are used to collect the initial data, so using a diverse set of policies and allowing human-guided exploration through "choice" policies (as we used in our study) helps the system generalize and consider a larger set of possible curricula.

In summary, when deploying a system similar to our own in a real world setting, a number of important design decisions need to be made, including the identification of facts, skills, dependency structures, proficiency adjustment rules, value ranges of nodes, and also what policies to use to collect the initial data. Often these choices involve a trade off between the granularity of the system and the amount of training data required.

While the choices above need to be made to deploy our current system in practical domains, there are also many extensions of the current system that warrant investigation. For instance, while we filled in DBN parameters using a "no change" heuristic in areas where the training data had no information, methods for active exploration of DBN parameters (Kearns and Koller 1999) could be used in a future system. However, these optimistic agents would try to explicitly gather data in areas where little information exists, such as teaching skills when most of their pre-requisites are are not yet mastered. Thus, there is an inherent tension between exploitation of the data from the expert policies and exploration of the parameter space. Two possibilities for mitigating this effect in the future are (1) training the system with software "virtual students" where exploring policies is far less costly than with human subjects and (2) using a more restricted form of exploration that does not deviate far from the collected data. Another possible extension is eliminating the assumption that a student's skill proficiency was essentially observable and based on a set of rules. Other ITS systems (Conati, Gertner, and Vanlehn 2002; Almond 2007) have looked instead at modeling a belief state over the student's proficiency. This would turn our representation into a POMDP and significantly complicate planning, but may result in better policies in the long run.

In this work we have shown how to build and train an ITS system represented as a factored sequential decision making model. Our human-subject trials show that this method can create unique policies that perform on par with hand-designed curricula.

Acknowledgements This work was supported by DARPA under contract HR 0011-09-C-0032.

References

Almond, R. G. 2007. Cognitive modeling to represent growth (learning) using markov decision processes. *Technology, Instruction, Cognition and Learning* 5: 313–324.

Barnes, T., and Stamper, J. 2008. Toward automatic hint generation for logic proof tutoring using historical student data. In *ITS*.

Beal, C.; Arroyo, I.; Cohen, P.; and Woolf, B. 2010. Evaluation of animalwatch: An intelligent tutoring system for arithmetic and fractions. *Journal of Interactive Online Learning* 9: 64–77.

Bloom, B. 1984. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher* 13: 4–16.

Conati, C.; Gertner, A.; and VanLehn, K. 2002. Using bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adapted Interaction* 12: 371–417.

Dean, T., and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Computational Intelligence* 5: 142–150.

Degris, T.; Sigaud, O.; and Wuillemin, P. H. 2006. Learning the structure of factored markov decision processes in reinforcement learning problems. In *ICML*.

Dekhtyar, A.; Goldsmith, J.; Goldstein, B.; Mathias, K. K.; and Isenhour, C. 2009. Planning for success: The interdisciplinary approach to building bayesian models. *Int. J. Approx. Reasoning* 50(3): 416–428.

Hoey, J.; St-Aubin, R.; Hu, A. J.; and Boutilier, C. 1999. Spudd: Stochastic planning using decision diagrams. In *UAI*.

Hoey, J.; Plötz, T.; Jackson, D.; Monk, A.; Pham, C.; and Olivier, P. 2010. Rapid specification and automated generation of prompting systems to assist people with dementia. *Pervasive and Mobile Computing* 1–29.

Kearns, M. J., and Koller, D. 1999. Efficient reinforcement learning in factored MDPs. In *IJCAI*.

Manske, M., and Conati, C. 2005. Modelling learning in an educational game. In *Artificial Intelligence in Education*.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* New York: Wiley.

Veermans, K.; van Joolingen, W.; and de Jong, T. 2006. Using heuristics to facilitate discovery learning in a simulation learning environment in a physics domain. *International Journal of Science Education* 21: 341–361.

Vygotsky, L. 1978. Interaction between learning and development. In *Mind In Society*. Cambridge, MA: Harvard University Press.