Acta Cybernetica **19** (2010) 635–653.

Modeling a Domain in a Tutorial-*like* System Using Learning Automata*

B. John Oommen† and M. Khaled Hashem‡

Abstract

The aim of this paper is to present a novel approach to model a knowledge domain for teaching material in a Tutorial-*like* system. In this approach, the Tutorial-*like* system is capable of presenting teaching material within a Socratic model of teaching. The corresponding questions are of a multiple choice type, in which the complexity of the material increases in difficulty. This enables the Tutorial-*like* system to present the teaching material in different chapters, where each chapter represents a level of difficulty that is harder than the previous one. We attempt to achieve the entire learning process using the Learning Automata (LA) paradigm. In order for the Domain model to possess an increased difficulty for the teaching Environment, we propose to correspondingly reduce the *range* of the penalty probabilities of all actions by incorporating a scaling factor μ. We show that such a scaling renders it more difficult for the Student to infer the correct action within the LA paradigm.

To the best of our knowledge, the concept of modeling teaching material with increasing difficulty using a LA paradigm is unique. The main results we have obtained are that increasing the difficulty of the teaching material can affect the learning of Normal and Below-Normal Students by resulting in an increased learning time, but it seems to have no effect on the learning behavior of Fast Students. The proposed representation has been tested for different benchmark Environments, and the results show that the difficulty of the Environments can be increased by decreasing the range of the penalty probabilities. For example, for some Environments, decreasing the range of the penalty probabilities by 50% results in increasing the difficulty of learning for Normal Students by more than 60%.

Keywords: domain modeling, tutorial-like systems, learning automata, mod-

*The first author was partially supported by NSERC, the Natural Sciences and Engineering Research Council of Canada. A preliminary version of this paper was presented at the Proceedings of ICMLC 2007, the 2007 International Conference of Machine Learning and Cybernetics, Hong Kong, China, August 2007.

†*Chancellor's Professor*; *Fellow: IEEE* and *Fellow: IAPR*. This author can be contacted at: School of Computer Science, Carleton University, Ottawa, Canada, K1S 5B6. This author is also an *Adjunct Professor* with the University of Agder in Grimstad, Norway. oommen@scs.carleton.ca.

‡This author can be contacted at: School of Computer Science, Carleton University, Ottawa, Canada, K1S 5B6. k_hashem@yahoo.com.

eling of adaptive systems.

1 Introduction

Representing the domain knowledge in Tutorials systems, in an effective way, is a challenging task. This includes how the knowledge is represented, and how it should be structured so as to reflect the nature of increasing the complexity/difficulty of the material to be taught in the Tutorial system [4]. The Domain model is, typically, the model that permits such a customization, and is usually application dependent.

From a systems perspective, the Domain model is the control center that encompasses the entire domain knowledge. The Teacher utilizes the domain knowledge as represented in the Domain model. Further, he[1] incorporates it into his teaching model to present the material to the Students in a manner that is customizable to each Student.

Analogous to traditional Tutorial systems, our Tutorial-*like* system utilizes the Domain model to represent the domain knowledge in a manner that enables the Teacher to conduct the learning to the Students in an effective way. The aim of this paper is to present how the domain knowledge can be modeled and implemented in such Tutorial-*like* systems. The domain knowledge is presented using a Socratic model and *via* multiple choice questions within the Learning Automata (LA) paradigm. For each question, every choice has an associated probability that this choice is correct, and the answer to any specific question is the choice with the highest reward probability.

Our model utilizes concepts from the field of LA, where the Domain model incorporates a novel mechanism to present the teaching material. The salient features of this mechanism can be summarized as follows:

- The knowledge is presented via multiple choice questions, which also serve to *test* the learning mechanism.
- The collection/set of questions also constitutes a chapter in the knowledge to be imparted.
- Subsequent chapters are more difficult than preceding ones.
- The answers to the question for subsequent chapters are not predictable by virtue of prior knowledge.

All of these details will be clarified presently.

In order for the Domain model to render a question to be more difficult, we propose that it reduces the so-called penalty probabilities for the choices pertinent to that question. This makes the Environment's response to the choices of that question less predictable. The experimental results of our model, as will be presented

[1] For the ease of communication, we request the permission to refer to the entities involved (i.e. the Teacher, Student, etc.) in the masculine.

later in the paper, demonstrate that this approach is feasible in representing the knowledge in a Tutorial-*like* system. The approach has been tested in benchmark Environments and the results are both intuitively appealing and rather fascinating considering that the Students and Teacher are not *real-life* entities, but rather, "models". Increasing the difficulty of the teaching environment proved to make the learning more difficult for Normal and Below-Normal Students, while Fast-learning Students were apparently not adversely effected by the increased difficulty. For example, for Below-Normal learners, when the range of the reward probabilities was decreased by 50% in the benchmark Environments, the difficulty within the teaching Environments increased by more than 60%. We believe that our Domain model representation is a novel approach within the field of LA modeling, which permits the LA Environments in this field to consistently increase their difficulties so as to mimic the teaching of material with increasing complexities.

The different components of Tutorial-*like* systems are also modeled, namely, the Student model, the Teacher model, and the Student-Classroom interaction model. These models have been studied in detail elsewhere [15, 14], but briefly mentioned here to permit readability.

1.1 Tutorial-*like* Systems

Our entire research will be within the context of Tutorial-*like* systems [14]. In these systems, there need not be *real-life* Students, but rather each Student could be replaced by a Student Simulator that mimics a *real-life* Student. Alternatively, it could also be a software entity that attempts to learn. The Teacher, in these systems, attempts to present the teaching material to a *School* of Student Simulators. The Students (synonymously referred to also as Student Simulators) are also permitted to share information between each other to gain knowledge. Therefore, such a teaching environment allows the Students to gain knowledge not only from the Teacher but also from other fellow Students.

In the Tutorial-*like* systems which we study, the Teacher has a *stochastic* nature, where he has an imprecise knowledge of the material to be imparted. The Teacher also doesn't have a prior knowledge about how to teach the subject material. He "learns" that himself while using the system, and thus, hopefully, improves his skills as a teacher. Observe that, conceptually, the Teacher, in some sense, is also a "student".

On the other hand, the Student Simulators need to learn from the Stochastic Teacher, as well as from each other. Each Student needs to decide when to request assistance from a fellow Student and how to "judge" the quality of information he receives from them. Thus, we require each Student to possess a mechanism whereby it can detect a scenario of procuring inaccurate information from other Students.

In our model of teaching/learning, the teaching material of the Tutorial-*like* system follows a Socratic model, where the domain knowledge is represented in the form of questions, either to be of a *Multiple Choice* sort or, in the most extreme case, of a *Boolean* sort. These questions, in our present paradigm, carry some degree of uncertainty, where each question has a probability that indicates the accuracy for

the answer of that question.

1.2 Stochastic Learning Automaton

Learning Automaton[2] (LA) have been used in systems that have incomplete knowledge about the Environment in which they operate [1, 17, 20, 21, 23, 30, 37]. The learning mechanism attempts to learn from a *stochastic Teacher* which models the Environment. In his pioneer work, Tsetlin [38] attempted to use LA to model biological learning. In general, a random action is selected based on a probability vector, and these action probabilities are updated based on the observation of the Environment's response, after which the procedure is repeated.

The term "Learning Automata" was first publicized in the survey paper by Narendra and Thathachar. The goal of LA is to "determine the optimal action out of a set of allowable actions" [1]. The distinguishing characteristic of automata-based learning is that the search for the optimizing parameter vector is conducted in the space of probability distributions defined over the parameter space, rather than in the parameter space itself [36].

In the first LA designs, the transition and the output functions were time invariant, and for this reason these LA were considered "fixed structure" automata. Tsetlin, Krylov, and Krinsky [38] presented notable examples of this type of automata. Later, Vorontsova and Varshavskii introduced a class of stochastic automata known in the literature as Variable Structure Stochastic Automata (VSSA). In the definition of a VSSA, the LA is completely defined by a set of actions (one of which is the output of the automaton), a set of inputs (which is usually the response of the Environment) and a learning algorithm, T. The learning algorithm [21] operates on a vector (called *the Action Probability vector*)

$\mathrm{P(t)} = [\mathrm{p_1(t)}, \ldots, \mathrm{p}_r\mathrm{(t)}]^{\mathrm{T}}$,

where $\mathrm{p}_i\mathrm{(t)}$ ($\mathrm{i} = 1, \ldots, \mathrm{r}$) is the probability that the automaton will select the action α_i at time 't',

$\mathrm{p}_i\mathrm{(t)} = \Pr[\alpha(\mathrm{t}) = \alpha_i]$, $\mathrm{i} = 1, \ldots, \mathrm{r}$, and it satisfies

$\sum_{i=1}^{r} \mathrm{p}_i\mathrm{(t)} = 1 \ \forall$ t.

Note that the algorithm $T : [0,1]^r \times \mathrm{A} \times \mathrm{B} \rightarrow [0,1]^r$ is an updating scheme where $\mathrm{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, $2 \leq \mathrm{r} < \infty$, is the set of output actions of the automaton, and B is the set of responses from the Environment. Thus, the updating is such that

$\mathrm{P(t+1)} = T(\mathrm{P(t)}, \alpha(\mathrm{t}), \beta(\mathrm{t}))$,

where P(t) is the action probability vector, α(t) is the action chosen at time t, and β(t) is the response it has obtained.

If the mapping T is chosen in such a manner that the Markov process has absorbing states, the algorithm is referred to as an absorbing algorithm. Many families of VSSA that posses absorbing barriers have been reported [21]. Ergodic VSSA have also been investigated [21, 25]. These VSSA converge in distribution and thus, the asymptotic distribution of the action probability vector has a value

[2] In the interest of completeness, we have included a fairly good review of the field of LA here. This can be deleted or abridged as per the desire of the Referees.

that is independent of the corresponding initial vector. Thus, while ergodic VSSA are suitable for non-stationary environments, automata with absorbing barriers are preferred in stationary environments.

In practice, the relatively slow rate of convergence of these algorithms constituted a limiting factor in their applicability. In order to increase their speed of convergence, the concept of discretizing the probability space was introduced [25, 35]. This concept is implemented by restricting the probability of choosing an action to a finite number of values in the interval [0,1]. If the values allowed are equally spaced in this interval, the discretization is said to be linear, otherwise, the discretization is called non-linear. Following the discretization concept, many of the continuous VSSA have been discretized; indeed, discrete versions of almost all continuous automata have been presented in the literature [25].

Pursuit and Estimator-based LA were introduced to be faster schemes, characterized by the fact that they pursue what can be reckoned to be the *current* optimal action or the set of current optimal schemes [25]. The updating algorithm improves its convergence results by using the history to maintain an estimate of the probability of each action being rewarded, in what is called the *reward-estimate* vector. While, in non-estimator algorithms, the action probability vector is updated solely on the basis of the Environment's response, in a Pursuit or Estimator-based LA, the update is based on *both* the Environment's response and the *reward-estimate* vector. Families of Pursuit and Estimator-based LA have been shown to be faster than VSSA [36]. Indeed, even faster discretized versions of these schemes have been reported [1, 25].

With regard to applications, the entire field of LA and stochastic learning, has had a myriad of applications [17, 20, 21, 30, 37], which (apart from the many applications listed in these books) include solutions for problems in network and communications [19, 22, 27, 29], network call admission, traffic control, quality of service routing, [2, 3, 40], distributed scheduling [34], training hidden Markov models [16], neural network adaptation [18], intelligent vehicle control [39], and even fairly theoretical problems such as graph partitioning [26]. Besides these fairly generic applications, with a little insight, LA can be used to assist in solving (by, indeed, learning the associated parameters) the stochastic resonance problem [10], the stochastic sampling problem in computer graphics [11], the problem of determining roads in aerial images by using geometric-stochastic models [6], the stochastic and dynamic vehicle routing problem [7], and various location problems [9]. Similar learning solutions can also be used to analyze the stochastic properties of the random waypoint mobility model in wireless communication networks [8], to achieve spatial point pattern analysis codes for GISs [31], to digitally simulate wind field velocities [28], to interrogate the experimental measurements of global dynamics in magneto-mechanical oscillators [12], and to analyze spatial point patterns [5]. LA-based schemes have already been utilized to learn the best parameters for neural networks [18], optimizing QoS routing [41], and bus arbitration [22] – to mention a few other applications.

1.3 Contributions of this paper

This paper presents a novel approach to model the domain knowledge in a Tutorial-*like* system. The representation of the knowledge can be crucial in building effective Tutorial systems. Thus, the salient contributions of this paper are as follows:

- The modeling of the domain knowledge using an LA paradigm using a Socratic model.
- Questions, in this model, are represented to be of a multiple-choice type, with stochastic solutions.
- The Student needs to learn the domain knowledge by responding to the questions.
- Enabling the Domain model to increase the complexity/difficulty of the domain knowledge.
- The knowledge is presented in chapters, where subsequent chapters are more difficult than preceding ones.

2 Intelligent Tutorial and Tutorial-*like* Systems

Since our research involves Tutorial-*like* systems, which are intended to mimic Tutorial systems, a brief overview of these follows.

Intelligent Tutorial Systems (ITSs) are special educational software packages that involve Artificial Intelligence (AI) techniques and methods to represent the knowledge, as well as to conduct the learning interaction [24]. ITSs are characterized by their responsiveness to the learner's need. They adapt according to the knowledge/skill of the users. They also incorporate experts' domain specific knowledge.

An ITS mainly consists of a number of modules, typically three [13], and sometimes four when a communication module (interface) is added [42]. The former three modules are the domain model (knowledge domain), the student model, and the pedagogical model, (which represent the tutor model itself). Self [33] defined these components as the tripartite architecture for an ITS – the *what* (domain model), the *who* (student model), and the *how* (tutoring model). Figure 1 depicts a common ITS architecture.

2.1 Tutorial-*like* Systems

Tutorial-*like* systems share some similarities with the well-developed field of Tutorial systems. Thus, for example, they model the Teacher, the Student, and the Domain knowledge. However, they are different from "traditional" Tutorial systems in the characteristics of their models, etc. as will be highlighted below.

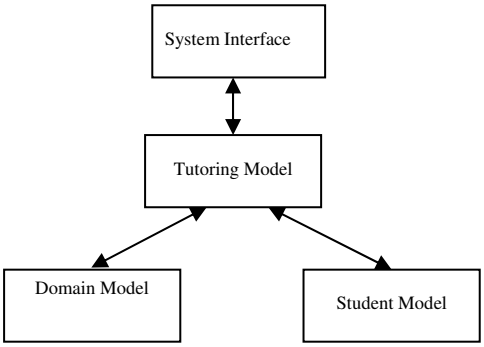

Figure 1: A Common ITS Architecture.

1. **Different Type of Teacher.** In Tutorial systems, as they are developed today, the Teacher is assumed to have perfect information about the material to be taught. Also, built into the model of the Teacher is the knowledge of how the domain material is to be taught, and a plan of how it will communicate and interact with the Student(s). This teaching strategy may progress and improve over time. The Teacher in our Tutorial-*like* system possesses different features. First of all, one fundamental difference is that the Teacher is uncertain of the teaching material – he is stochastic. Secondly, the Teacher does not *initially* possess any knowledge about "How to teach" the domain subject. Rather, the Teacher himself is involved in a "learning" process and he "learns" what teaching material has to be presented to the particular Student. To achieve this, as mentioned, we assume that the Teacher follows the Socratic model of learning by teaching the material using questions that are presented to the Students. He then uses the feedback from the Students and their corresponding LA to suggest new teaching material.

   Although removing the "How to teach" knowledge from the Teacher would take away the "bread and butter" premise of the teaching process in a Tutorial system, in a Tutorial-*like* system, removing this knowledge allows the system to be modeled without excessive complications, and renders the modeling of knowledge less burdensome. The success of our proposed methodology would be beneficial to systems in which any domain knowledge pertinent to tutoring teaching material could be merely plugged into the system without the need to worry about "how to teach" the material.

2. **No Real Students.** A Tutorial system is intended for the use of *real-life* students. Its measure of accomplishment is based on the performance of these students after using the system, and it is often quantified by comparing their progress with other students in a control group, who would use a *real-life* Tutor. In our Tutorial-*like* system, there are no *real-life* students who use the system. The system could be used by either:

a) Students Simulators, that mimic the behavior and actions of *real-life* students using the system. The latter would themselves simulate how the Students improve their knowledge and their interaction with the Teacher and with other Students. They can also take proactive actions interacting with the teaching environment by one of the following measures:
   i. Asking a question to the Teacher
   ii. Asking a question to another Student
   iii. Proposing to help another Student

b) An artificial Entity which, in itself, could be another software component that needs to "learn" specific domain knowledge.

3. **Uncertain Course Material.** Unlike the domain knowledge of "traditional" Tutorial systems where the knowledge is, typically, well defined, the domain knowledge teaching material presented in our Tutorial-*like* system contains material that has some degree of uncertainty. The teaching material contains questions, each of which has a probability that indicates the certainty of whether the answer to the question is in the affirmative.

4. **Testing Vs. Evaluation.** Sanders [32] differentiates between the concepts of "teaching evaluation" and "teaching testing". He defines "teaching evaluation" as an "interpretive process", in which the Teacher "values, determines merit or worth of the students performance, and their needs". He also defines "teaching testing" as a "data collection process". In a Tutorial system, an evaluation is required to measure the performance of the Student while using the system and acquiring more knowledge. In our Tutorial-*like* system, the Student(s) acquire knowledge using a Socratic model, where it gains knowledge from answering questions without having any prior knowledge about the subject material. In our model, the testing will be based on the performance of the set of Student Simulators.

5. *School* **of Students.** Traditional Tutorial Systems deal with a Teacher who teaches Students, but they do not permit the Students to interact with each other. A Tutorial-*like* system assumes that the Teacher is dealing with a *School* of Students where each learns from the Teacher on his own, and can also learn from his "colleagues" if he desires, or is communicating with a cooperating colleague. Notice that we will have to now consider how each Student learns, and also how the entire *School* learns.

3 Learning of Students in a LA Teaching Environment

In Tutorial-*like* systems, Students (or Student Simulators) try to learn some domain knowledge from the Teacher and from the interaction between themselves. As mentioned earlier, there are no *real-life* Students who use the Tutorial-*like* systems.

Students are modeled using Student Simulators, that try to mimic the actions and behavior of *real-life* Students. Student Simulators are, in turn, modeled using LA which attempt to learn the domain knowledge from the Teacher, who also may be a modeled entity.

First of all, the Tutorial-*like* system models the Students by observing their behavior while using the system and examining how they learn. The Student modeler tries to infer what *type* of Student it is providing the knowledge to. This enables the Teacher to customize his teaching experience to each Student according to his caliber.

If we are dealing with *real-life* Students, it would have been an easy task to implement these concepts in a real Tutorial system. But since the goal of the exercise is to achieve a teaching-learning experience, in which every facet of the interaction involves a model (or a non *real-life* Student), the design and implementation of the entire Tutorial-*like* system must be couched in a formal established learning paradigm. As mentioned earlier, although there are host of such learning methodologies, we have chosen to work within the LA paradigm, as explained in Section 1.2. Thus, the questions encountered, before this endeavor is successful, involve:

- How can we model the Teacher in terms of an LA Environment?
- How can we model the different types of Students that could be involved in the learning?
- How can we model the Domain, from which the learning material is presented?
- How can we model *chapters* of teaching material with increasing complexity?

We shall address all of these issues now, and report the experimental results obtained from such a modeling exercise.

Modeling The Student: In our model, typically, a Student can be one of these three types (although it is easy to generalize this to a larger spectrum of Students):

- Fast Student. This type of Students can be simulated using a Pursuit scheme, which is, typically, a fast convergence scheme.
- Normal Student. The Student Simulator can mimic this type of Students using a VSSA scheme.
- Slow Student. Such a Student can be implemented using a FSSA, or a VSSA with a lower value of λ.

Modeling the Choices: The Tutorial-*like* system uses the Socratic model of teaching by presenting multiple-choice questions to the Students. The Student selects an option from the set of available actions $\underline{\alpha}$, in which α_i is the action corresponding to selecting choice 'i' in the multiple-choice question.

Modeling the stochastic Teacher: The Teacher who imparts the domain knowledge is modeled as an LA Environment, which possesses a set of penalty

probabilities $\underline{c}$, in which c_i is the penalty probability associated with the fact that the Environment penalizes choice 'i'. The Student is unaware of the values of these penalty probabilities.

Modeling the Rewards/Penalties: When the Student selects an action α_i, the Environment can either reward ($\beta = 0$) or penalize ($\beta = 1$) his actions. This feedback provides the Student the information required to learn from his actions, and from this feedback loop, the cycle is repeated. The Student can incrementally learn until his LA, hopefully, converges to the *best* action, which is the one which has the minimum penalty probability.

The crucial issue that has not been addressed as yet is that of modeling the domain itself to consider "chapters" of increasing complexity. That will be addressed and formalized in the next section.

4 Domainsem/Teaching Material with Increasing Difficulty

The Teaching material in the Tutorial-*like* system is modeled as a Socratic model that contains multiple choice questions. Each choice has an associated probability that represents the probability that this choice is correct. Each question is represented using an LA Environment, where the Environment has a penalty probability associated with that choice.

When the Domain model is required to increase the complexity of a question, we propose that it reduces the penalty[3] probabilities of the choices for that question with a scale factor, μ. This results in the reduction of the range of all the penalty probabilities, which makes it more difficult for the Student to determine the *best* choice for the question, primarily because the reduced penalty probabilities tend to cluster together. This is the primary hypothesis of our model, and this will be demonstrated presently.

Formally, the Domain model for the teaching Environment is as follows:

$\{\underline{\alpha}, \underline{\beta}, \underline{c}, \mu\}$, where:

$\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_R\}$, in which

α_i is the action corresponding to selecting choice 'i' in the multiple choice question.

$\underline{\beta} = \{0, 1\}$, in which

$\beta = 0$ implies a Reward for the present action (i.e, choice) chosen, and

$\beta = 1$ implies a Penalty for the present action (i.e, choice) chosen.

$\underline{c} = \{c_1, c_2, \ldots, c_R\}$, in which

c_i is the penalty probability associated with the fact that the Environment will penalize choice 'i'.

μ ($0 < \mu \leq 1$) is the scaling factor which is used to control the complexity/difficulty of any question. The value of μ=1.0 represents a question with a "normal" difficulty, while the difficulty increases as μ decreases.

[3] It is easy to see that a similar scaling can be achieved by manipulating the *reward* probabilities.

The Domain model increases the complexity of the domain knowledge without changing the order of the choices to the question. If the Student is permitted to remember the choices of previous questions, he is indirectly given prior knowledge about the optimal answer to the particular question at hand. However, for the present we assume that as far as the Student is concerned, the chapter presented by the Environment is not related to a previous one[4].

5 Experimental Results

This section presents the experimental results obtained by using the proposed Domain model to represent the teaching material, and to increase its complexity. Numerous simulations were performed in order to study how the knowledge could be modeled, and how the difficulty of the teaching material led to increase the learning time for the different types of Students.

The simulations were performed for different benchmark Environments, two 4-action Environments and two 10-action Environments. The Environments contained multiple choice questions that represented the teaching material that had to be taught to the Students. In these experiments, an algorithm was considered to have converged, if the probability of choosing an action was greater than or equal to a threshold $T(0 < T \leq 1)$. An automaton was considered to converge correctly if it converged to the best choice (i.e. to the action with the highest probability of being rewarded).

The simulations were performed against three types of Students, who communicated with the Teacher and learned the teaching material as follows:

- Fast learning Students. In order for the Student Simulator to mimic Students of this type, we used a Pursuit PL_{RI} scheme, with λ being in the range 0.0041 to 0.0127. In this scheme, the action probability vector is updated if the LA obtained a reward. As for the Pursuit algorithm, the estimate vector for the reward probabilities was always updated.

- Normal learning Students. In this case, we used a VSSA to simulate Students of this type. In particular, we utilized the L_{RI} scheme with λ being in the range 0.0182 to 0.0192.

- Below-Normal Students ("Slow Learners"). In this case too, the Student Simulators used VSSA to simulate learners of this type. Again, our model used the L_{RI} scheme, but with a lower value of λ, which was between 0.0142 to 0.0152.

[4]The question of how to deal with chapters of increasing complexity is dealt with elsewhere [14]. Without belaboring the point, we mention that in order for the Domain model to keep the identity of superior actions in subsequent chapters hidden from the Student, in [14] we propose to shuffle the order of the choices that are presented to the Student. Therefore, the Student can not use his prior knowledge to "short-cut" an answer to more difficult learning material.

The simulations were performed for the different 4-action and 10-action benchmark Environments, for which the threshold T was set to be 0.99, and the number of experiments (NE) = 75. The results of these simulations are described below.

5.1 Results using 4-action and 10 action Environments

The teaching Environments in the simulations have been represented by two benchmark sets of Environments. The first set includes two 4-action Environments ($E_{4,A}$ and $E_{4,B}$), and the second set contains two 10-action benchmark Environments ($E_{10,A}$ and $E_{10,B}$). The 4-action Environment represents a multiple-choice question with 4 options, whereas the 10-action Environment represents a more difficult multiple-choice question, namely with 10 options. The Students in the simulations needed to learn the responses for the questions and determine the choice that possessed the minimum penalty probability.
For $E_{4,A}$ and $E_{10,A}$, the λ of the Student Simulators LA were set to be:

- 0.0127 for the Fast learning Student.
- 0.0192 for the Normal learning Student.
- 0.0142 for the Below-Normal learning Student.

Also, for $E_{4,B}$ and $E_{10,B}$, the λ of the Student Simulators LA were:

- 0.0041 for the Fast learning Student.
- 0.0182 for the Normal learning Student.
- 0.0152 for the Below-Normal learning Student.

For the 4-action Environments, the reward probabilities were:
$E_{4,A} = \{0.7 \quad 0.5 \quad 0.3 \quad 0.2\}$ and
$E_{4,B} = \{0.1 \quad 0.45 \quad 0.84 \quad 0.76\}$.
Similarly, for the 10-action Environments, the two settings for the reward probabilities were:
$E_{10,A} = \{0.7 \quad 0.5 \quad 0.3 \quad 0.2 \quad 0.4 \quad 0.5 \quad 0.4 \quad 0.3 \quad 0.5 \quad 0.2\}$ and
$E_{10,B} = \{0.1 \quad 0.45 \quad 0.84 \quad 0.76 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.7 \quad 0.5 \quad 0.3\}$.
The simulations were performed for the different Environments and types of Students, as described above. Also, the experiments were conducted by controlling the Domain model with different factors of difficulty μ, from the range of 1.0 (no difficulty) to 0.3. The results of these simulations are provided in Table 1.

The results demonstrate that the difficulty of the Domain knowledge increased by decreasing μ for both the Normal and Below-Normal learners. As opposed to this, Fast learners were apparently not adversely affected by the increasing difficulty of the Domain knowledge. For example, in the $E_{4,A}$ Environment, a Normal learner Student LA converged in 975 iterations to learn the material in an Environment without any enhanced level of difficulty (μ=1.0), while it converged in 1,474 iterations when μ decreased to 0.6, which represents an increase of 51% in

Env.	**No. of actions**	Chapter	μ (diffic. factor)	No. iterations for **Fast Learner** to converge	No. iterations for **Normal Learner** to converge	No. iterations for **Below Norm. Learner** to converge
		1	1.0	563	975	1,380
		2	0.9	542	1,051	1,497
		3	0.8	530	1,192	1,628
$\mathbf{E_{4,A}}$	4	4	0.7	515	1,321	1,722
		5	0.6	506	1,474	2,085
		6	0.5	473	1,682	2,245
		7	0.4	492	1,918	3,077
		8	0.3	523	2,393	3,512
		1	1.0	1,480	2,046	2,459
		2	0.9	1,420	2,326	2,799
		3	0.8	1,388	2,247	2,894
$\mathbf{E_{4,B}}$	4	4	0.7	1,445	2,500	3,525
		5	0.6	1,443	2,631	3,533
		6	0.5	1,378	2,897	3,887
		7	0.4	1,445	3,569	4,652
		8	0.3	1,407	3,651	5,036
		1	1.0	680	1,320	1,728
		2	0.9	684	1,442	1,972
		3	0.8	660	1,485	2,256
$\mathbf{E_{10,A}}$	10	4	0.7	667	1,731	2,386
		5	0.6	641	1,963	2,845
		6	0.5	650	2,226	3,424
		7	0.4	656	2,604	3,965
		8	0.3	711	3,105	5,085
		1	1.0	1,631	2,286	2,773
		2	0.9	1,623	2,282	2,662
		3	0.8	1,580	2,608	3,319
$\mathbf{E_{10,B}}$	10	4	0.7	1,555	2,879	3,644
		5	0.6	1,590	2,928	3,580
		6	0.5	1,527	3,478	4,460
		7	0.4	1,644	3,717	4,628
		8	0.3	1,715	4,015	5,776

Reward probabilities for 4-action Environments are:

$\mathbf{E_{4,A}}$: 0.7 0.5 0.3 0.2

$\mathbf{E_{4,B}}$: 0.1 0.45 0.84 0.76

Reward probabilities for 10-action Environments are:

$\mathbf{E_{10,A}}$: 0.7 0.5 0.3 0.2 0.4 0.5 0.4 0.3 0.5 0.2

$\mathbf{E_{10,B}}$: 0.1 0.45 0.84 0.76 0.2 0.4 0.6 0.7 0.5 0.3

Table 1: Convergence of the Student Simulators learning in the benchmark Four/Ten-Action Environments by increasing the level of difficulty in the Domain knowledge.

the learning time for the Student. When the difficulty increased further by setting μ=0.3, the iterations needed for learning increased to 2,393, which represents an increase of 145% of the time required to learn when compared to the original benchmark Environment.

Similar results were also recorded for the Below-Normal learner, where he learned the material in 1,380 iteration in an Environment without any added difficulty. When the difficulty of the material increased corresponding to a value of μ=0.6, the number of iterations increased to 2,085, which represents a 51% increase in the learning time. The learning time increased to 3,512 iterations when the control parameter μ was 0.3, which represents a 154% increase in the learning time from the original benchmark.

On the other hand, the results showed that Fast learners were not adversely affected by increasing the difficulty in the Environment. Indeed, the number of iterations needed to learn the teaching material were almost constant. This seems to also be the case for *real-life* Students.

Similar results are also observed for the other Environments ($E_{4,B}$, $E_{10,A}$, and $E_{10,B}$), as seen from Table 1.

Figure 2 depicts graphically the results of the simulations. For each benchmark Environment, it displays the relationship between the number of iterations and μ. The reader should observe the apparent "proportional" increase in the number of iterations by increasing the complexity (i.e. by decreasing μ) for both Normal and Below-Normal learners. It also shows that Fast learners were not affected by the increased complexity of the problem.

6 Conclusion

This paper introduced a novel approach of modeling the Domain knowledge in a Tutorial-*like* system. In this model, the Domain knowledge is presented in different chapters, where the difficulty of the learned knowledge increased with the subsequent chapters.

The Domain knowledge has been modeled using the concept of Environments in a LA paradigm, from which the Student Simulator LA are trying to learn. The model presented in the paper showed that the difficulty of the Domain knowledge (as increasingly more complex chapters were encountered) could be increased by decreasing the range of the penalty probabilities of all the pertinent actions by multiplying them with a factor, μ. The main results that we have obtained is that the learning time increased for Normal and Below-Normal learners as the difficulty of the Domain knowledge increased. This was not the case for Fast learners, which seems to be consistent with our experience with *real-life* Students.

The Teacher will be using the Domain model to present the teaching material in a chapter-wise fashion to the Students. He will need to determine when the chapter complexity can be increased, and how prior knowledge can be used by the Student LA in such Tutorial-*like* systems. This is currently being done elsewhere [14].

For future work, we are considering how we can use this approach to model

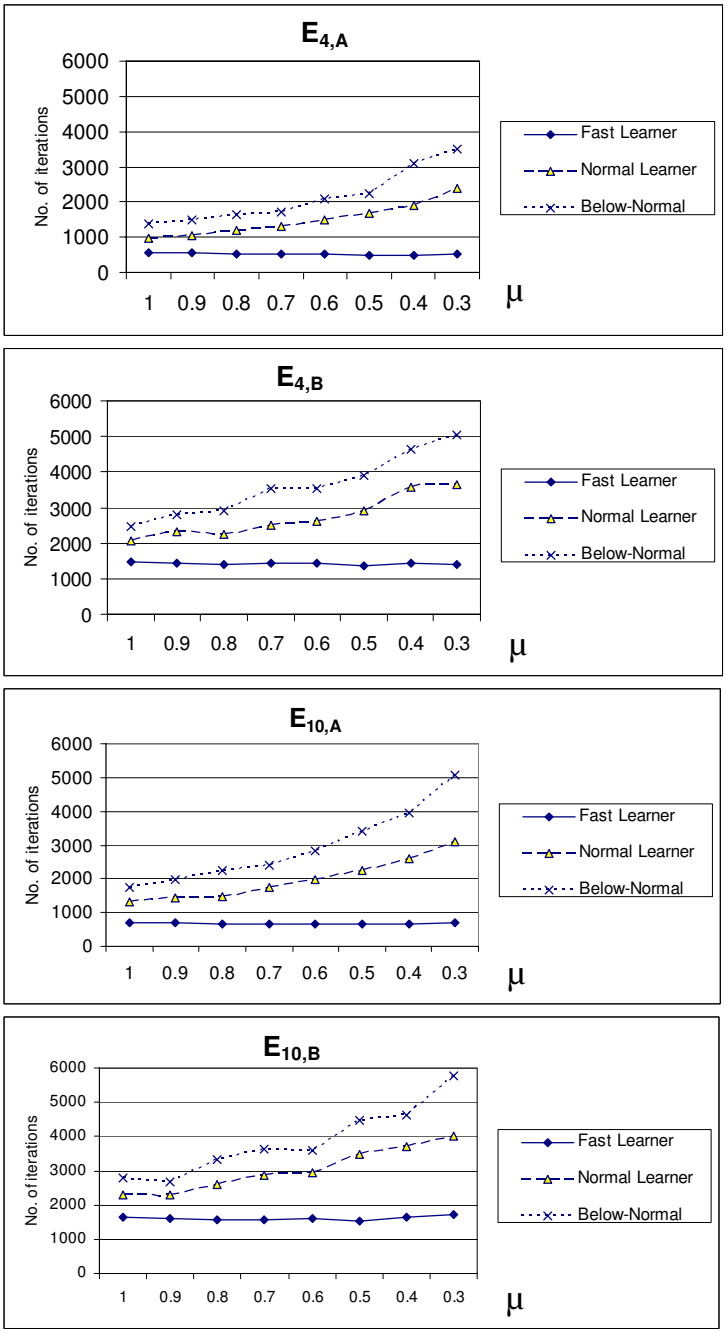

Figure 2: The effect of increasing the difficulty of the Domain model on the different types of Students in the four benchmark Environments.

the increasing complexity of *real-life* domain knowledge, where the multiple choice questions that the Student answers are from a real domain. A more distant future work is to port this approach to be used in traditional Tutorial systems, where the Students can be taught uncertain domain knowledge.

References

[1] Agache, M. and Oommen, B. J. Generalized pursuit learning schemes: New families of continuous and discretized learning automata. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 32(6):738–749, December 2002.

[2] Atlassis, A. F., Loukas, N. H., and Vasilakos, A. V. The use of learning algorithms in atm networks call admission control problem: A methodology. *Computer Networks*, 34:341–353, 2000.

[3] Atlassis, A. F. and Vasilakos, A. V. The use of reinforcement learning algorithms in traffic control of high speed networks. *Advances in Computational Intelligence and Learning*, pages 353–369, 2002.

[4] Atolagbe, T. A. and Hlupic, V. SimTutor: A multimedia intelligent tutoring system for simulation modeling. In Andraddttir, S., Healy, K. J., Withers, D. H., and Nelson, B. L., editors, *Proceedings of the 29th Conference on Winter Simulation*, pages 504–509, Atlanta, Georgia, 1997.

[5] Baddeley, A. and Turner, R. Spatstat: An R package for analyzing spatial point patterns. *Journal of Statistical Software*, 12:1–42, 2005.

[6] Barzohar, M. and Cooper, D. B. Automatic finding of main roads in aerial images by using geometric-stochastic models and estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7:707–722, 1996.

[7] Bertsimas, D. J. and Ryzin, G. Van. Stochastic and Dynamic vehicle routing in the euclidean plane with multiple capacitated vehicles. *Operations Research*, 41:60–76, 1993.

[8] Bettstetter, C., Hartenstein, H., and Pérez-Costa, X. Stochastic properties of the random waypoint mobility model. *Journal Wireless Networks*, 10:555–567, 2004.

[9] Brandeau, M. L. and Chiu, S. S. An overview of representative problems in Location Research. *Management Science*, 35:645–674, 1989.

[10] Collins, J. J., Chow, C. C., and Imhoff, T. T. Aperiodic stochastic resonance in excitable systems. *Physical Review E*, 52:R3321–R3324, 1995.

[11] Cook, R. L. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5:51–72, 1986.

[12] Cusumano, J. P. and Kimble, B. W. A stochastic interrogation method for experimental measurements of global dynamics and basin evolution: Application to a two-well oscillator. *Nonlinear Dynamics*, 8:213–235, 1995.

[13] Fischetti, E. and Gisolfi, A. From computer-aided instruction to intelligent tutoring systems. *Educational Technology*, 30(8):7–17, 1990.

[14] Hashem, M. K. *Learning Automata Based Intelligent Tutorial-like Systems.* PhD thesis, School of Computer Science, Carleton University, Ottawa, Canada, 2007.

[15] Hashem, M. K. and Oommen, B. J. On using learning automata to model a student's behavior in a tutorial-*like* system. In *Proceedings of the IEA/AIE 2007: The 20th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems*, pages 813–822, Kyoto, Japan, June 2007.

[16] Kabudian, J., Meybodi, M. R., and Homayounpour, M. M. Applying continuous action reinforcement learning automata (CARLA) to global training of hidden markov models. In *Proceedings of the International Conference on Information Technology: Coding and Computing , ITCC'04*, pages 638–642, Las Vegas, Nevada, 2004.

[17] Lakshmivarahan, S. *Learning Algorithms Theory and Applications.* Springer-Verlag, 1981.

[18] Meybodi, M. R. and Beigy, H. New learning automata based algorithms for adaptation of backpropagation algorithm pararmeters. *International Journal of Neural Systems*, 12:45–67, 2002.

[19] Misra, S. and Oommen, B. J. GPSPA: A new adaptive algorithm for maintaining shortest path routing trees in stochastic networks. *International Journal of Communication Systems*, 17:963–984, 2004.

[20] Najim, K. and Poznyak, A. S. *Learning Automata: Theory and Applications.* Pergamon Press, Oxford, 1994.

[21] Narendra, K. S. and Thathachar, M. A. L. *Learning Automata: An Introduction.* Prentice-Hall, New Jersey, 1989.

[22] Obaidat, M. S., Papadimitriou, G. I., Pomportsis, A. S., and Laskaridis, H. S. Learning automata-based bus arbitration for shared-medium ATM switches. *IEEE Transactions on Systems, Man, and Cybernetics: Part B*, 32:815–820, 2002.

[23] Obaidat, M. S., Papadimitrious, G. I., and Pomportsis, A. S. Learning automata: Theory, paradigms, and applications. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 32(6):706–709, December 2002.

[24] Omar, N. and Leite, A. S. The learning process mediated by intelligent tutoring systems and conceptual learning. In *International Conference On Engineering Education*, page 20, Rio de Janeiro, 1998.

[25] Oommen, B. J. and Agache, M. Continuous and discretized pursuit learning schemes: Various algorithms and their comparison. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 31:277–287, 2001.

[26] Oommen, B. J. and Croix, E.V. de St. Graph partitioning using learning automata. *IEEE Transactions on Computers*, C-45:195–208, 1995.

[27] Oommen, B. J. and Roberts, T. D. Continuous learning automata solutions to the capacity assignment problem. *IEEE Transactions on Computers*, C-49:608–620, 2000.

[28] Paola, M. Digital simulation of wind field velocity. *Journal of Wind Engineering and Industrial Aerodynamics*, 74-76:91–109, 1998.

[29] Papadimitriou, G. I. and Pomportsis, A. S. Learning-automata-based TDMA protocols for broadcast communication systems with bursty traffic. *IEEE Communication Letters*, pages 107–109, 2000.

[30] Poznyak, A. S. and Najim, K. *Learning Automata and Stochastic Optimization*. Springer-Verlag, Berlin, 1997.

[31] Rowlingson, B. S. and Diggle, P. J. *SPLANCS: spatial point pattern analysis code in S-Plus*. University of Lancaster, North West Regional Research Laboratory, 1991.

[32] Sanders, J. R. In *Twenty-fourth Annual Meeting of The Joint Committee on Standards for Educational Evaluation*, October 1998.

[33] Self, J. The defining characteristics of intelligent tutoring systems research: ITSs care, precisely. *International Journal of AI in Education*, 10:350–364, 1999.

[34] Seredynski, F. Distributed scheduling using simple learning machines. *European Journal of Operational Research*, 107:401–413, 1998.

[35] Thathachar, M. A. L. and Oommen, B. J. Discretized reward-inaction learning automata. *Journal of Cybernetics and Information Science*, pages 24–29, Spring 1979.

[36] Thathachar, M. A. L. and Sastry, P. S. Varieties of learning automata: An overview. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 32(6):711–722, December 2002.

[37] Thathachar, M. A. L. and Sastry, P. S. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic, Boston, 2003.

[38] Tsetlin, M. L. *Automaton Theory and the Modeling of Biological Systems*. Academic Press, New York, 1973.

[39] Unsal, C., Kachroo, P., and Bay, J. S. Simulation study of multiple intelligent vehicle control using stochastic learning automata. *Transactions of the Society for Computer Simulation International*, 14:193–210, 1997.

[40] Vasilakos, A., Saltouros, M. P., Atlassis, A. F., and Pedrycz, W. Optimizing QoS routing in hierarchical ATM networks using computational intelligence techniques. *IEEE Transactions on Systems Science, and Cybernetics, Part C*, 33:297–312, August 2003.

[41] Vasilakos, A. V., Saltouros, M. P., Atlassis, A. F., and Pedrycz, W. Optimizing QoS routing in hierarchical ATM networks using computational intelligence techniques. *IEEE Transactions on Systems, Man, and Cybernetics: Part C*, 33:297–312, 2003.

[42] Winkels, R. and Breuker, J. What's in an ITS? a functional decomposition. In Costa, E., editor, *New Directions for Intelligent Tutoring Systems*. Spring-Verlag, Berlin, 1990.

Received 18th July 2008