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

B. John Oommen[⋆,⋆⋆] and M. Khaled Hashem

School of Computer Science, Carleton University, Ottawa, Canada, K1S 5B6

Abstract. The goal of this paper is to present a novel approach to model the behavior of a *Teacher* in a Tutorial-*like* system. In this model, the Teacher is capable of presenting teaching material from a Socratic-type Domain model *via* multiple-choice questions. Since this knowledge is stored in the Domain model in chapters with different levels of complexity, the Teacher is able to present learning material of varying degrees of difficulty to the Students.

In our model, we propose that the Teacher will be able to assist the Students to learn the more difficult material. In order to achieve this, he provides them with *hints* that are relative to the difficulty of the learning material presented. This enables the Students to cope with the process of handling more complex knowledge, and to be able to learn it appropriately.

To our knowledge, the findings of this study are novel to the field of intelligent adaptation using Learning Automata (LA). The novelty lies in the fact that the learning system has a strategy by which it can deal with increasingly more complex/difficult Environments (or domains from which the learning as to be achieved). In our approach, the convergence of the Student models (represented by LA) is driven not only by the response of the Environment (Teacher), but also by the *hints* that are provided by the latter. Our proposed Teacher model has been tested against different benchmark Environments, and the results of these simulations have demonstrated the salient aspects of our model. The main conclusion is that Normal and Below-Normal learners benefited significantly from the hints provided by the Teacher, while the benefits to (brilliant) Fast learners were marginal. This seems to be in-line with our subjective understanding of the behavior of *real-life* Students.

Keywords: Tutorial-like Systems, Learning Automata, Modeling of Adaptive Tutorial Systems, Modeling of Teacher.

1 Introduction

The Teacher is the main source of knowledge to Students in a teaching environment. The success of the Students to effectively learn the domain knowledge is

[⋆] Partially supported by NSERC, the Natural Sciences and Engineering Research Council of Canada. A preliminary version of this paper was presented at the IEEE-SMC 2007, the 2007 IEEE International Conference on Systems, Man and Cybernetics, Montreal, Canada, October 2007.

[⋆⋆] *Chancellor's Professor*; *Fellow: IEEE* and *Fellow: IAPR*. He is also an *Adjunct Professor* with the University of Agder in Grimstad, Norway.

mainly influenced by the ability and skills of the Teacher. Similarly, the effectiveness of a Tutorial-*like* system is influenced by the modeling of the Teacher, so that the knowledge would be imparted successfully to the Students.

In a Tutorial system, the Teacher model is a representation of the Teacher and for the process whereby he[1] takes pedagogical decisions to communicate and impart the teaching material to the Students. It decides on what, how and when the material must be presented to the Students.

In our Tutorial-*like* system, the Teacher is modeled to teach a Socratic-type Domain model, where the knowledge is represented *via* multiple-choice questions, which are used also to *test* the Students in the knowledge imparted. The aim of this paper is present, within the Learning Automata (LA) paradigm, a new approach where the Teacher not only presents a Socratic-type domain knowledge to the Students, but he also *assists* them so that they would be able to learn more complex material. The Teacher tasks in imparting the knowledge involves the following concepts:

- He obtains knowledge from the Domain model to present it to the Students.
- The Domain model contains knowledge, of Socratic-type presented *via* multiple-choice questions.
- The domain knowledge is modeled with increasing complexity, which is intended to represent the increasing complexity of subsequent chapters. Thus, each chapter represents a level of difficulty that is harder than the previous one.
- Students learn from the questions presented to them, and by the Teacher testing them in these questions.
- The Teacher is capable of improving the abilities of the Students to learn more complex material by assisting them to handle this augmented difficulty.

Briefly, we achieve this goal as follows. In order for the Teacher to enable the Students to improve their abilities to handle more complex material, he provides them with *hints*. The Teacher provides these hints in the form of suggestions to the Students so as to increase the initial probability for one of the actions in the action probability vector, P. The Teacher provides this hint to the correct answer, which is the action with the highest reward probability, in a stochastic manner, which, in turn, assigns the probability that the Student, correctly, receives the *hint*. In our model, that probability increases with the increasing complexity of the material being taught.

The experimental results of this model, as will be presented later, confirm that our approach is feasible in modeling the Teacher in a Tutorial-*like* system. The proposed model have been tested against different benchmark Environments, and the results have shown that our model was successful in teaching the Students, and in assisting them to improve their abilities to learn even as the domain become increasingly complex. The main finding from the simulations is that Normal and Below-Normal Students succeeded significantly in improving their abilities to learn more complex domain knowledge when the Teacher provided them with these

[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.

hints. For example, in some Environments, the learning of a Normal Student improved by more than 67% when the Teacher provided *hints* to the Students, when the initial action probability of the best action was increased by 0.6.

To our knowledge, this represents the first published results in which the learning of the LA, in terms of Student Simulators, is influenced not only by the response from the Teacher (i.e. an Environment within a LA paradigm), but also by the *hints* that are provided by the Teacher.

1.1 Tutorial-*like* Systems

Our entire research will be within the context of Tutorial-*like* systems [1]. 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 [2,3,4,5,6,7,8]. The learning mechanism attempts to learn from a *stochastic Teacher* which models the Environment. In his pioneer work, Tsetlin [9] attempted to use LA to model biological learning. In general, a random action is selected based on a probability

[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.

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" [2]. 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 [10].

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 [9] 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 [5] operates on a vector (called *the Action Probability vector*)

$P(t) = [p_1(t), \ldots, p_r(t)]^T$,

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

$p_i(t) = \Pr[\alpha(t) = \alpha_i]$, $i = 1, \ldots, r$, and it satisfies

$\sum_{i=1}^{r} p_i(t) = 1 \ \forall\ t$.

Note that the algorithm $T : [0,1]^r \times A \times B \rightarrow [0,1]^r$ is an updating scheme where $A = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, $2 \leq r < \infty$, 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

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

where $P(t)$ is the action probability vector, $\alpha(t)$ is the action chosen at time t, and $\beta(t)$ 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 [5]. Ergodic VSSA have also been investigated [5,11]. These VSSA converge in distribution and thus, the asymptotic distribution of the action probability vector has a value 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 [11,15]. 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 [11].

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 [11]. 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 [10]. Indeed, even faster discretized versions of these schemes have been reported [2,11].

With regard to applications, the entire field of LA and stochastic learning, has had a myriad of applications [3,4,5,7,8], which (apart from the many applications listed in these books) include solutions for problems in network and communications [16,17,18,19], network call admission, traffic control, quality of service routing, [20,21,22], distributed scheduling [23], training hidden Markov models [24], neural network adaptation [25], intelligent vehicle control [26], and even fairly theoretical problems such as graph partitioning [27]. 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 [28], the stochastic sampling problem in computer graphics [29], the problem of determining roads in aerial images by using geometric-stochastic models [30], the stochastic and dynamic vehicle routing problem [31], and various location problems [32]. Similar learning solutions can also be used to analyze the stochastic properties of the random waypoint mobility model in wireless communication networks [33], to achieve spatial point pattern analysis codes for GISs [34], to digitally simulate wind field velocities [35], to interrogate the experimental measurements of global dynamics in magneto-mechanical oscillators [36], and to analyze spatial point patterns [37]. LA-based schemes have already been utilized to learn the best parameters for neural networks [25], optimizing QoS routing [38], and bus arbitration [17] – to mention a few other applications.

1.3 Contributions of This Paper

This paper presents a novel approach for modeling how a Teacher can present and teach Socratic-type domain knowledge, with varying degrees of difficulty, to the Students, or Student Simulators. In this model, the Teacher, apart from "teaching", will also assist Students to handle more complex domain knowledge material. Thus, the salient contributions of this paper are:

– The modeling of a Teacher in a Tutorial-*like* system, within the LA paradigm.
– The Teacher who interacts with Students, provides them with material from the domain knowledge, and responds to their answers for the questions.
– The concept of a Teacher providing *hints* to the Students so as to assist them in handling more complex information.

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 [39]. 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 [40], and sometimes four when a communication module (interface) is added [41]. The former three modules are the domain model (knowledge domain), the student model, and the pedagogical model, (which represent the tutor model itself). Self [42] 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.

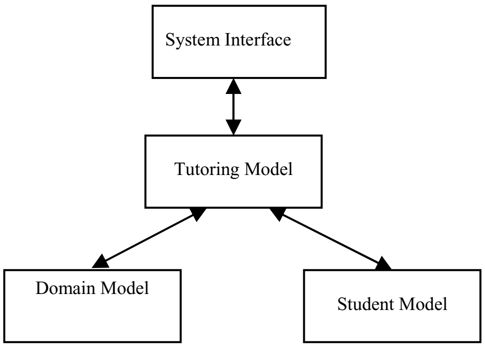

Fig. 1. A Common ITS Architecture

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.

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 [43] 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 Concept of Teachers Who Provide *Hints*

3.1 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?
- How can we model the process of the Teacher assisting the Student to learn the more complex material?

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) [1,12]:

- 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.

The details of this modeling process is explained in [1,12]. The students themselves work collectively in a Classroom, as explained in [14], the details of which are omitted here.

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 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.

Modeling the Domain with increasing complexity: When the Domain model is required to increase the complexity of a question, we proposed in [1] the strategy by which the penalty probabilities of the choices for that question are reduced by 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.The details of this modeling process is explained in detail in [1,13].

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.

The only critical issue that has not been addressed as yet is that of how the Teacher can assist the Students to learn more complex material *via* the so-called *hints*. This will be addressed and formalized in the next section.

The issue of how the Teacher *himself* can learn is currently being compiled for publication.

3.2 Model for Teachers Who Can Provide *Hints*

As mentioned earlier, the Teacher, in our Tutorial-*like* system, is modeled to use a Socratic-type Domain model. The domain knowledge is represented using multiple-choice questions, with chapters of contents with increasing complexity.

When the Teacher provides the Students with more complex material, we are left with the problem of him also possessing the ability to provide them with the means to deal with this increased complexity. We resolve this by proposing that he presents them with *hints*, so that they can improve their learning abilities and cope with the complexity of the domain knowledge.

Modeling a Teacher with these capabilities, who can teach and also assist the Students and provide them with *hints*, can be formally defined as:

$\{\underline{\alpha}, \underline{\beta}, \underline{c}, \mu\ \rho, \mathrm{P}_{init}, \sigma_B\}$, where:

- $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_R\}$, in which
  α_i is the action corresponding to selecting choice 'i' in the question.
- $\underline{\beta} = \{0, 1\}$, in which
  $\beta = 0$ implies a Reward for the present action (i.e, choice) chosen by the Student, 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 penalizes 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. μ will also be referred to as the *difficulty factor* (or index).
- ρ $(0 \leq \rho < 1)$ is the *hint* value, which is used to control the extent of assistance which the Teacher provides to the Students. The value of ρ=0 represents the scenario when there is no enhanced assistance, while the value of the hint increases with ρ. ρ is also referred to as the *hint factor* (or index).
- P_{init} is the initial value of the Student's action probability vector, which contains the probabilities that the Student assigns to each of *his* actions. For the Student's action probability vector, at each instant t, the Student Simulator's LA randomly selects an action $\alpha(t) = \alpha_i$ with probability $p_i(t) = \Pr[\alpha(t) = \alpha_i]$, where $\sum_{i=1}^{r} p_i(t) = 1$. It is this vector which is initialized by P_{init}, which, in turn, is related to ρ.
  Without any hints from the Teacher, if the Student has R choices (actions) to choose from, the Student Simulator's LA, initially, will have an equal probability for each choice. Therefore, the action probability vector will be:
  $[\frac{1}{R}\ \ \frac{1}{R}\ \ \cdots]^{\mathrm{T}}$.
  However, when the Teacher provides a *hint* to the Student, we propose that these initial values change as per the value of the hint, ρ, as follows:
  - For action 'j' to which the Teacher provides the advantageous hints to the Student:
    $\mathrm{p}_j = \mathrm{p}_j + \rho.(\frac{R-1}{R})$.

- For all other actions 'i', where $i \neq j$:

  $p_i = p_i - \varDelta$, where $\varDelta = \frac{1}{R-1}\left(\frac{\rho(R-1)}{R}\right) = \frac{\rho}{R}$.

– σ_B is the probability that the Teacher will provide a *hint* to the Student indicating the identity of the *Best* action.

All of these concepts are explained in the following example.

3.3 Example of Teachers Who Can Provide *Hints*

Consider the case when the domain knowledge is represented by a 4-action Environment, in which the penalty probabilities are represented by:

$\underline{c} = \{0.3 \quad 0.1 \quad 0.7 \quad 0.5\}$

Note that in this Environment, the correct action is α_2, which possesses the minimum penalty probability. The initial values for the Student's action probability vector *without any hints* from the Teacher is:

$P(0) = [0.25 \quad 0.25 \quad 0.25 \quad 0.25]^T$.

If the Teacher wants to convey the information that he strategically believe that α_2 is the superior action, he provides the Students with a *hint* ρ, which has a value of, say, 0.4.

In this case, the action probability vector could be one of the following 4 options:

$P(0) = [0.55 \quad 0.15 \quad 0.15 \quad 0.15]^T$.
$P(0) = [0.15 \quad 0.55 \quad 0.15 \quad 0.15]^T$.
$P(0) = [0.15 \quad 0.15 \quad 0.55 \quad 0.15]^T$.
$P(0) = [0.15 \quad 0.15 \quad 0.15 \quad 0.55]^T$.

The probability that the Student will receive the *hint* in favor of α_2 is σ_B, while the probability that the Student will receive it for any other action is $\left(\frac{1-\sigma_B}{R-1}\right)$. If, for instance, σ_B is selected to be equal to p_B, then, in this example, the probability that the Student will receive the *hint* in favor of α_2 is 0.55, while the probability that he will receive it in favor of any other action is 0.15.

4 Experimental Results

In this section, we present the experimental results obtained by testing our Teacher model that provides *hints* to the Students. The results were obtained by performing numerous experiments to simulate the interaction between the Teacher and the Student Simulators, and the strategy by which the Teacher can affect the learning of the Students.

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

As mentioned earlier, three types of Students have been used in the simulations, who communicated with the Teacher to learn the subject matter as follows:

- Fast learning Students. To mimic this type of Students, the Student Simulator used a Pursuit PL$_{RI}$ scheme, with λ being in the range 0.0041 to 0.0127. The action probability vector for this scheme was updated only if the Student Simulator's LA obtained a reward. The estimate vector, however, for the reward probabilities was always updated.
- Normal learning Students. To simulate Students of this type, the Student Simulators used VSSA. In particular, it utilized the L$_{RI}$ scheme with λ being in the range 0.0182 to 0.0192.
- Below-Normal learning Students ("Slow Learners"). The Student Simulators also 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.

4.1 Teaching Domain Model with Increasing Difficulty, with *hints*

The Domain model in the simulations used domain knowledge with increasing difficulty. The simulations first used domain knowledge with no enhanced difficulty (μ=1.0). Thereafter, it used a more difficult domain, obtained by lower values of μ. In particular, we report the results for μ=0.8, 0.6, and 0.4.

For each level of difficulty in the Domain model, the Teacher communicated with the Students to teach them the learning material. The Teacher presented the domain knowledge to the Students in the following steps:

- First, he provided the knowledge to the Students with no enhanced assistance.
- Thereafter, he started providing assistance to the Students, with a *hint factor* of ρ.
- The value of the *hint factor* ρ ranged from 0.1 (for marginal hints), and increased to 0.8 to allow maximal assistance.
- When the Teacher provided a *hint* to the Student, it affected the Student Simulator's *initial* action probability vector, as defined earlier in Section 3.2.
- The probability that the Teacher would provide a *hint* to the Student (indicating the identity of the *Best* action), σ_B, increased with the values of ρ. In our simulations, for simplicity, σ_B was always set to be equal to p$_B$ (the initial probability of selecting the *Best* action in the corresponding action probability vector).

4.2 Results Using 4-Action and 10 Action Environments

The simulation experiments were performed using two sets of benchmark Environments, two 4-action Environments ($E_{4,A}$ and $E_{4,B}$), and two 10-action Environments ($E_{10,A}$ and $E_{10,B}$). While the 4-action Environment represents a multiple-choice question with 4 options, the 10-action Environment represents

a more difficult multiple-choice question with 10 options. The three different types of Students were assigned to learn the responses for the questions, and to determine the *best* choice for each question, which is the one that possesses the minimum penalty probability.
For the different types of the Student Simulators, the λ for $E_{4,A}$ and $E_{10,A}$ were:

- 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 set to be:

- 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 two settings for 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 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 results of these simulations are tabulated in Tables 1-4 for Environments $E_{4,A}$, $E_{4,B}$, $E_{10,A}$, and $E_{10,B}$ respectively.

$E_{4,A}$ Environment. The results for the $E_{4,A}$ Environment are given in Table 1. The results show that when the Teacher provided *hints* to Normal and Below-Normal Students, their learning improved significantly. For example, in a domain characterized by the *difficulty factor* $\mu = 0.8$, the learning of a Normal Student improved by 58% when the Teacher provided *hints* with $\rho = 0.8$, compared with the learning without any hints, as the number of iterations required for learning decreased from 1,273 to 541. Similarly, if $\mu = 0.6$, a Below-Normal Student improved his learning when the Teacher provided *hints* with $\rho = 0.8$, by 47% as the Student learned the material in 1,107 iterations instead of 2,086 iterations.

Although, in general, the rate of learning improved for Normal and Below-Normal Students with the increase of ρ, the improvement was marginal with small ρ, and significant when ρ ¿ 0.3. This can be attributed to the fact that, in our experiments, σ_B was selected to be equal to p_B. With smaller ρ, σ_B was also small.

However, Fast Students showed less advantage in their learning when the Teacher provided them with *hints*. For example, in a domain with $\mu = 0.4$, while the number of iterations required for learning without any hints was 479, it decreased to 388 when the Teacher provided *hints* with $\rho = 0.8$, which represents only a 19% learning improvement.

The results of the $E_{4,A}$ Environment simulations are depicted graphically in Figure 2. For a domain knowledge with increasing complexity that ranges from μ=1.0 (no enhanced difficulty) until $\mu = 0.4$, we see that, in general, the number of iterations needed for learning decreased with the increase of ρ. This, of course, is also intuitively appealing.

$E_{4,B}$ Environment. Although the $E_{4,B}$ Environment was more difficult than the $E_{4,A}$ Environment, the results obtained for it were similar to the latter. Normal and Below-Normal Students showed a noticeable improvement when provided with *hints* from the Teacher, while the learning gain for Fast Students was marginal. The results for the $E_{4,B}$ Environment are given in Table 2.

For example, in a Domain with a *difficulty index* $\mu = 0.8$, the learning of the Normal Student improved to require only 761 iterations when the *hint factor* ρ was 0.8, instead of 2,330 iteration when no hints were provided. This represents a 67% improvement. The learning for Normal Students even exceeds that of Fast Students when ρ was large (more than 0.7). Similarly, in an Environment with $\mu = 0.6$, the Below-Normal Student improved his skills by learning in 1,218 iterations (for $\rho = 0.8$), instead of 3,517 iterations, which was the time required for learning without any enhanced assistance. This implies a 65% improvement in the learning.

As before, the gain for Fast Students was minimal. For example, when $\mu = 0.4$, a Fast Student improved his learning by only 12% when the Teacher provided *hints* with $\rho = 0.8$. The number of iterations decreased from 1,379 (with no hints) to 1,213.

The results for the $E_{4,B}$ Environment simulations are shown graphically in Figure 3. The figure shows the improvement in the Student learning with the increase of ρ for different values of μ, the complexity of the Domain model. The reader must observe the reverse proportionality relationship between the *hint index* ρ, and the number of iterations needed for the learning for the different types of Students.

$E_{10,A}$ Environment. In the intent of completeness, we now briefly report the results for a single 10-action Environment, $E_{10,A}$, which represents a more difficult Environment than the 4-action Environment. The simulation results obtained for this Environment are essentially similar to the ones obtained for the 4-action Environments. The gains for the Normal and Below-Normal Students were substantial, while the gains for Fast Students were marginal. The results for the $E_{10,A}$ Environment are given in Table 3.

We mention a few specific examples here. In a domain knowledge with a *difficulty index* $\mu = 0.6$, the Normal Student improved his abilities by learning the domain knowledge in 818 iterations (with $\rho = 0.7$), as opposed to 1,960 iterations, with no enhanced assistance from the Teacher. This represents a 58% improvement in the learning. Also, for the Below-Normal Student in an

Table 1. Convergence of the Student Simulators when they are learning in the benchmark $\mathbf{E_{4,A}}$ Environment with increasing *Difficulty Indices*, and increasing *Hint Indices*. In all these cases, σ_B was assigned the same value as p_B (as per the notation of Section 3.3).

μ (diffic. fact.)	ρ (hint)	$p_B = \sigma_B$	No. iterations for **Fast Learner** to converge	No. iterations for **Normal Learner** to converge	No. iterations for **Below Norm. Learner** to converge
1.0	0.0	0.250	560	941	1,383
	0.1	0.325	548	978	1,375
	0.2	0.400	550	990	1,459
	0.3	0.475	549	982	1,348
	0.4	0.550	534	829	1,166
	0.5	0.625	504	960	1,196
	0.6	0.700	478	763	1,077
	0.7	0.775	463	629	888
	0.8	0.850	396	516	653
0.8	0.0	0.250	532	1,273	1,645
	0.1	0.325	504	1,161	1,608
	0.2	0.400	521	1,104	1,674
	0.3	0.475	489	1,267	1,547
	0.4	0.550	493	1,052	1,536
	0.5	0.625	480	952	1,295
	0.6	0.700	456	916	1,106
	0.7	0.700	428	772	1,053
	0.8	0.850	371	541	861
0.6	0.0	0.250	500	1,427	2,086
	0.1	0.325	483	1,562	2,196
	0.2	0.400	487	1,438	1,870
	0.3	0.475	474	1,418	1,971
	0.4	0.550	468	1,357	1,818
	0.5	0.625	441	1,188	1,732
	0.6	0.700	457	1,062	1,571
	0.7	0.775	447	932	1,383
	0.8	0.850	390	801	1,107
0.4	0.0	0.250	479	1,996	2,792
	0.1	0.325	472	1,854	3,138
	0.2	0.400	475	2,032	3,074
	0.3	0.475	465	1,886	2,781
	0.4	0.550	476	1,728	2,560
	0.5	0.625	450	1,708	2,176
	0.6	0.700	444	1,514	1,760
	0.7	0.775	436	1,145	1,685
	0.8	0.850	388	894	1,135

Reward probabilities for $\mathbf{E_{4,A}}$ Environment are:
$\mathbf{E_{4,A}}$: 0.7 0.5 0.3 0.2

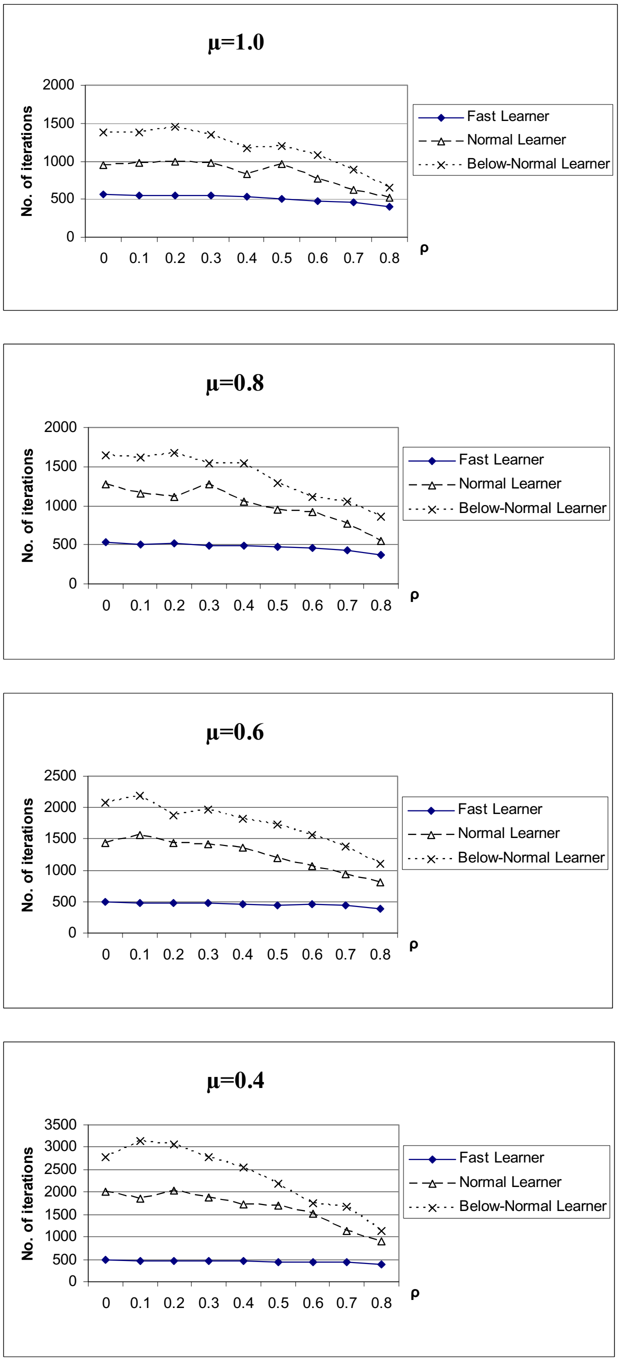

Fig. 2. The effect of increasing the *Hint Index* provided by the Teacher on Students learning for Environments derived from $E_{4,A}$ by varying the *Difficulty Indices*

Table 2. Convergence of the Student Simulators when they are learning in the benchmark $\mathbf{E_{4,B}}$ Environment with increasing *Difficulty Indices*, and increasing *Hint Indices*. In all these cases, σ_B was assigned the same value as p_B (as per the notation of Section 3.3).

μ (diffic. fact.)	ρ (hint)	$p_B = \sigma_B$	No. iterations for **Fast Learner** to converge	No. iterations for **Normal Learner** to converge	No. iterations for **Below Norm. Learner** to converge
	0.0	0.250	1,466	2,103	2,699
	0.1	0.325	1,472	1,979	2,780
	0.2	0.400	1,484	2,005	2,500
	0.3	0.475	1,449	1,880	2,269
1.0	0.4	0.550	1,403	1,622	2,137
	0.5	0.625	1,326	1,379	1,997
	0.6	0.700	1,281	1,427	1,666
	0.7	0.775	1,188	1,109	1,416
	0.8	0.850	1,130	762	1,076
	0.0	0.250	1,406	2,330	2,826
	0.1	0.325	1,419	2,556	3,162
	0.2	0.400	1,356	2,333	2,954
	0.3	0.475	1,346	2,259	2,585
0.8	0.4	0.550	1,335	2,452	2,738
	0.5	0.625	1,348	1,723	2,209
	0.6	0.700	1,259	1,711	1,896
	0.7	0.700	1,185	1,291	1,650
	0.8	0.850	1,259	761	1,397
	0.0	0.250	1,450	2,728	3,517
	0.1	0.325	1,352	2,872	3,717
	0.2	0.400	1,347	2,740	3,560
	0.3	0.475	1,328	2,671	3,528
0.6	0.4	0.550	1,303	2,039	3,103
	0.5	0.625	1,336	1,902	2,843
	0.6	0.700	1,293	1,522	2,304
	0.7	0.775	1,166	1,412	2,003
	0.8	0.850	1,153	838	1,218
	0.0	0.250	1,379	2,869	4,061
	0.1	0.325	1,477	3,275	4,125
	0.2	0.400	1,383	3,579	3,838
	0.3	0.475	1,282	2,837	3,646
0.4	0.4	0.550	1,325	3,386	3,925
	0.5	0.625	1,338	1,932	2,767
	0.6	0.700	1,299	1,847	2,541
	0.7	0.775	1,311	1,391	2,203
	0.8	0.850	1,213	1,017	1,403
Reward probabilities for $\mathbf{E_{4,B}}$ Environment are: $\mathbf{E_{4,B}}$: 0.1 0.45 0.84 0.76					

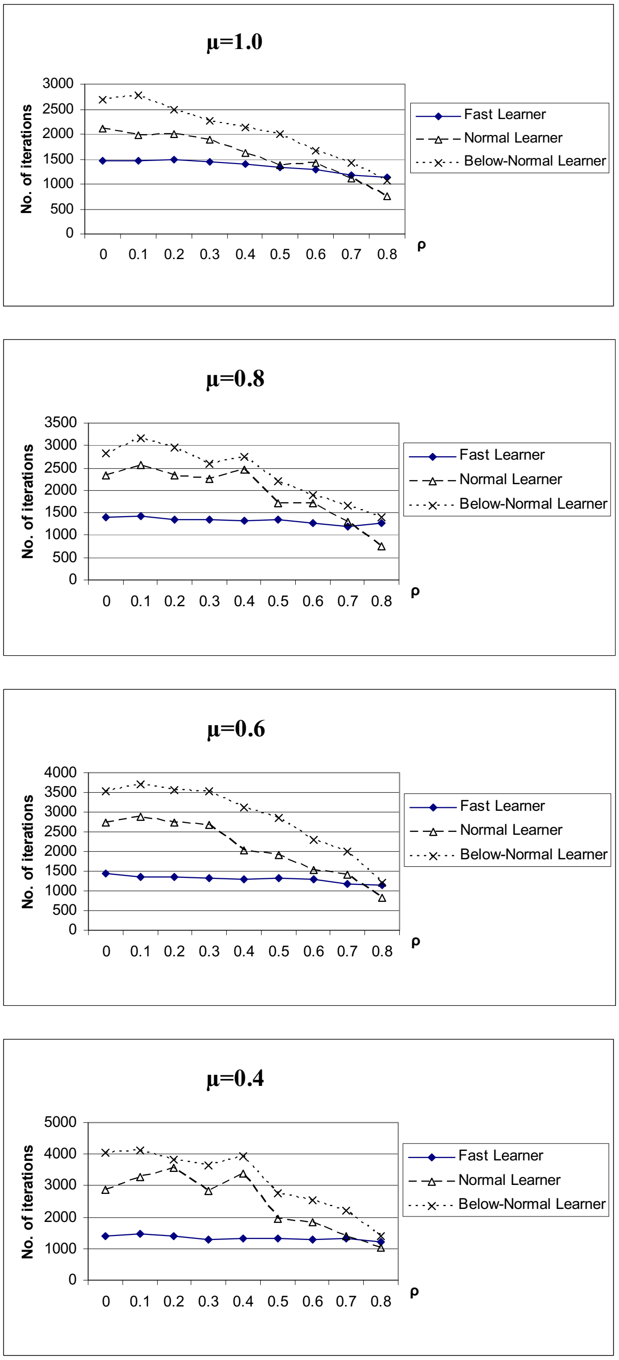

Fig. 3. The effect of increasing the *Hint Index* provided by the Teacher on Students learning for Environments derived from $E_{4,B}$ by varying the *Difficulty Indices*

Table 3. Convergence of the Student Simulators when they are learning in the benchmark $\mathbf{E_{10,A}}$ Environment with increasing *Difficulty Indices*, and increasing *Hint Indices*. In all these cases, σ_B was assigned the same value as p_B (as per the notation of Section 3.3).

μ (diffic. fact.)	ρ (hint)	$p_B = \sigma_B$	No. iterations for **Fast Learner** to converge	No. iterations for **Normal Learner** to converge	No. iterations for **Below Norm. Learner** to converge
1.0	0.0	0.10	633	1,367	1,802
	0.1	0.19	640	1,307	1,796
	0.2	0.28	619	1,221	1,799
	0.3	0.37	625	1,214	1,667
	0.4	0.46	628	1,126	1,643
	0.5	0.55	576	1,061	1,488
	0.6	0.64	571	9,55	1,275
	0.7	0.73	560	830	1,199
	0.8	0.82	501	597	842
0.8	0.0	0.10	574	1,525	2,219
	0.1	0.19	615	1,510	2,184
	0.2	0.28	607	1,458	2,200
	0.3	0.37	575	1,424	2,147
	0.4	0.46	607	1,302	1,738
	0.5	0.55	572	1,233	1,767
	0.6	0.64	556	1,030	1,469
	0.7	0.73	545	1,035	1,152
	0.8	0.82	522	660	890
0.6	0.0	0.10	613	1,960	2,722
	0.1	0.19	595	1,957	2,820
	0.2	0.28	608	1,824	2,757
	0.3	0.37	596	1,777	2,726
	0.4	0.46	570	1,692	2,289
	0.5	0.55	615	1,471	2,279
	0.6	0.64	590	1,253	1,633
	0.7	0.73	549	818	1,484
	0.8	0.82	578	691	1,181
0.4	0.0	0.10	627	2,827	3,813
	0.1	0.19	649	2,591	3,873
	0.2	0.28	622	2,250	3,685
	0.3	0.37	651	2,148	3,558
	0.4	0.46	631	1,998	2,955
	0.5	0.55	640	1,739	2,864
	0.6	0.64	617	1,530	2,333
	0.7	0.73	620	1,345	1,691
	0.8	0.82	591	660	1,256

Reward probabilities for $\mathbf{E_{10,A}}$ Environment 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

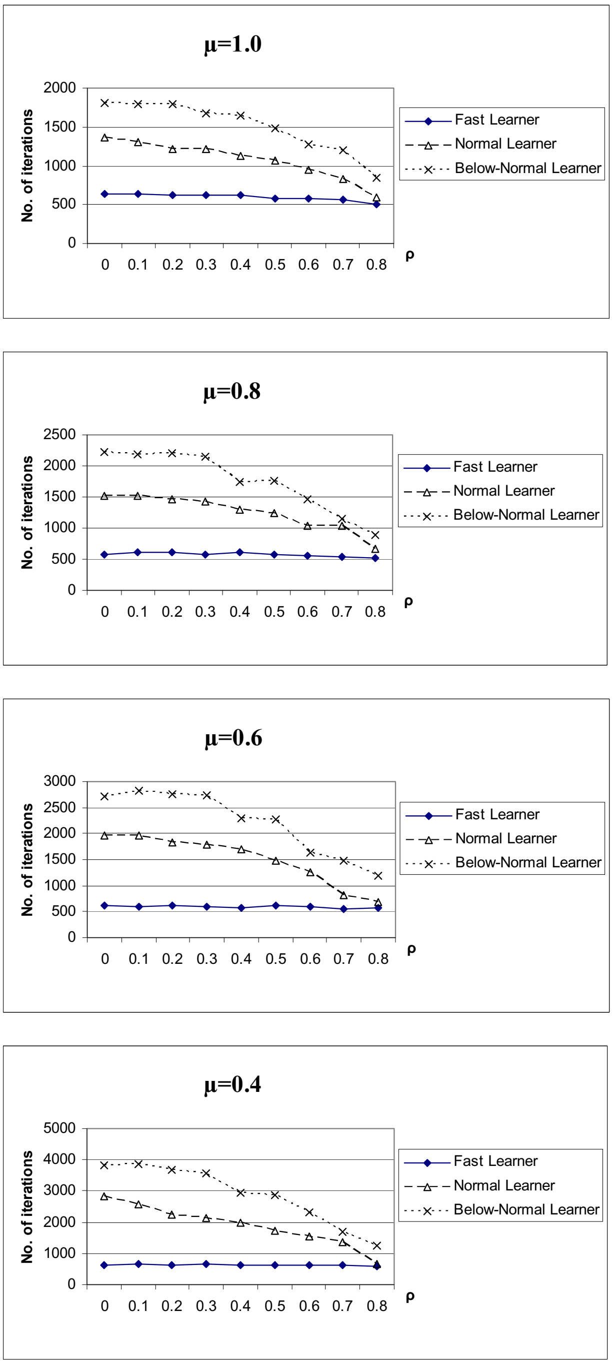

Fig. 4. The effect of increasing the *Hint Index* provided by the Teacher on Students learning for Environments derived from $E_{10,A}$ by varying the *Difficulty Indices*

Table 4. Convergence of the Student Simulators when they are learning in the benchmark $\mathbf{E_{10,B}}$ Environment with increasing *Difficulty Indices*, and increasing *Hint Indices*. In all these cases, σ_B was assigned the same value as p_B (as per the notation of Section 3.3).

μ (diffic. fact.)	ρ (hint)	$p_B = \sigma_B$	No. iterations for **Fast Learner** to converge	No. iterations for **Normal Learner** to converge	No. iterations for **Below Norm. Learner** to converge
1.0	0.0	0.10	1,615	2,198	2,748
	0.1	0.19	1,606	2,291	2,775
	0.2	0.28	1,585	2,157	2,554
	0.3	0.37	1,558	2,065	2,359
	0.4	0.46	1,517	1,674	2,126
	0.5	0.55	1,497	1,454	2,104
	0.6	0.64	1,459	1,280	1,848
	0.7	0.73	1,396	942	1,473
	0.8	0.82	1,319	504	1,069
0.8	0.0	0.10	1,610	2,589	3,372
	0.1	0.19	1,545	2,448	2,857
	0.2	0.28	1,576	2,299	2,763
	0.3	0.37	1,512	2,121	2,733
	0.4	0.46	1,510	1,833	2,699
	0.5	0.55	1,477	1,718	2,293
	0.6	0.64	1,448	1,219	1,391
	0.7	0.73	1,404	914	1,345
	0.8	0.82	1,252	736	883
0.6	0.0	0.10	1,583	3,280	3,857
	0.1	0.19	1,480	3,213	3,455
	0.2	0.28	1,550	2,441	3,656
	0.3	0.37	1,524	2,245	3,068
	0.4	0.46	1,503	1,893	2,772
	0.5	0.55	1,453	2,038	2,562
	0.6	0.64	1,407	1,637	2,068
	0.7	0.73	1,392	1,055	1,396
	0.8	0.82	1,351	843	1,101
0.4	0.0	0.10	1,640	3,935	5,024
	0.1	0.19	1,576	3,161	4,784
	0.2	0.28	1,569	3,137	4,237
	0.3	0.37	1,522	2,995	3,715
	0.4	0.46	1,595	2,397	2,889
	0.5	0.55	1,520	2,087	2,821
	0.6	0.64	1,559	1,629	2,348
	0.7	0.73	1,407	1,399	2,210
	0.8	0.82	1,330	1,083	1,252

Reward probabilities for $\mathbf{E_{10,B}}$ Environment are:

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

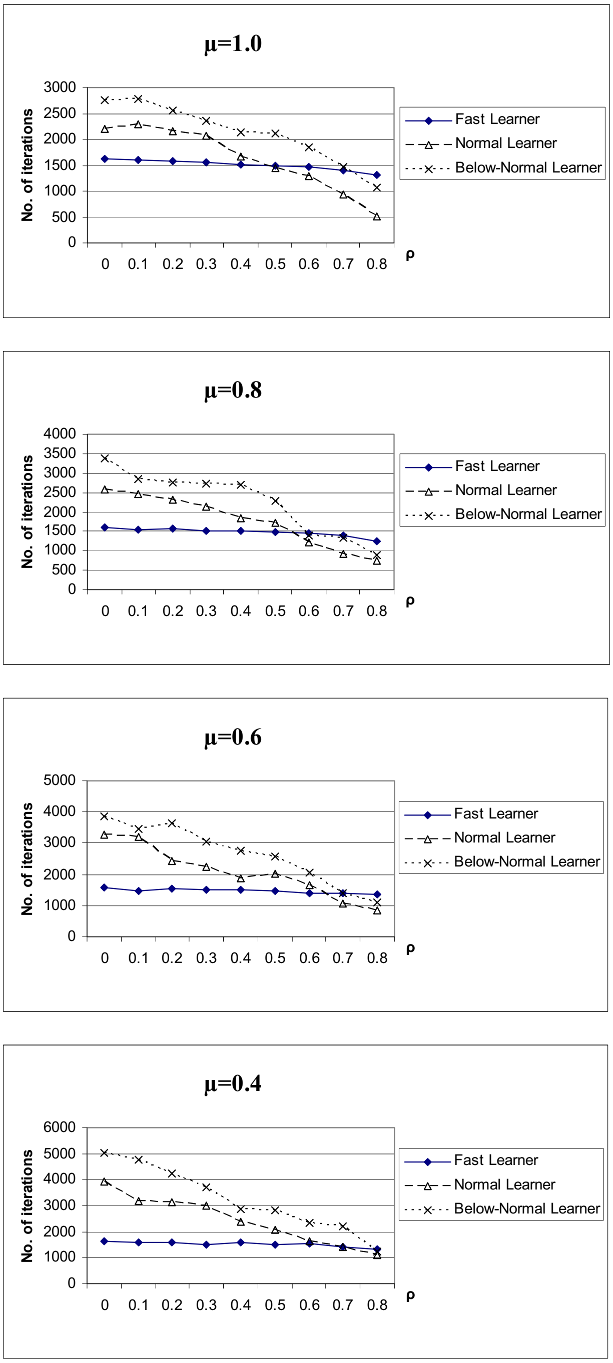

Fig. 5. The effect of increasing the *Hint Index* provided by the Teacher on Students learning for Environments derived from $E_{10,B}$ by varying the *Difficulty Indices*

Environment with the *difficulty index* $\mu = 0.4$, the Student learned the material in 1,691 iterations when the *hint factor* ρ was 0.7, compared to 3,813 iterations when there were no hints. This is a 55% improvement in the learning.

For Fast Students, the improvement in the learning was again marginal. For example, in an Environment with *difficulty index* $\mu = 0.4$, the improvement in learning attained only 6% when the Teacher provided *hints* with $\rho = 0.8$. The number of iterations required for the learning decreased from 627 (with no hints from the Teacher) to 591 iterations.

The results of the $\mathrm{E}_{10,\mathrm{A}}$ Environment simulations are depicted graphically in Figure 4. The figure shows that the learning of the Students improved with the increase of the *hint factor* ρ. The improvement is apparent for the Normal and Below-Normal Students, while it is rather insignificant for Fast Students.

Similar results were also observed for the $\mathrm{E}_{10,\mathrm{B}}$ Environment. The simulation results of this Environment are tabulated in Table 4 and depicted graphically in Figure 5. As before, the figure shows the reverse proportionality relationship between the value of the *hint index* ρ, and the number of iterations required for Students to learn the domain knowledge.

5 Conclusion and Future Work

This paper presented a novel paradigm to model the behavior of a Teacher in a Tutorial-*like* system. In this model, the Teacher is capable of presenting, to the Students, a Socratic-type Domain model in the form of multiple-choice question. As the Domain model is capable of storing domain knowledge with increasing levels of complexity, the Teacher is able to present these material to the Students.

In our proposed model, the Teacher can assist the Students so as to be able to improve their abilities to learn more complex knowledge. He provides them with *hints* via so-called "hint factors" or indices. The value of the *hint factor* is relative to the complexity of the domain knowledge that is presented. The main result obtained is that Normal and Below-Normal Students benefited substantially from *hints* provided by the Teacher. However, the benefits to Fast Students were not so significant. These results seem to fit with our view of how *real-life* Students respond to the assistance they receive from the Teacher.

We also foresee a strategy by which the Teacher *himself* will be able to "learn" so as to improve his "teaching" skills, and customize his teaching based on the learning capability of the corresponding Student. This is currently being investigated.

For a longer-term future goal, we would like to believe that our approach of providing *hints* can also be ported for use in real-life "traditional" Tutorial Systems. By allowing the Teacher model to present *hints* to *real-life* Students, they can, hopefully, be expected to benefit and learn more complex material. But this clearly, needs much more research, and collaborative involvement with scientists who work with both cognitive and realistic models of learning and teaching.

References

1. Hashem, M.K.: Learning Automata Based Intelligent Tutorial-like Systems. PhD thesis, School of Computer Science, Carleton University, Ottawa, Canada (2007)
2. Agache, M., 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 (2002)
3. Lakshmivarahan, S.: Learning Algorithms Theory and Applications. Springer (1981)
4. Najim, K., Poznyak, A.S.: Learning Automata: Theory and Applications. Pergamon Press, Oxford (1994)
5. Narendra, K.S., Thathachar, M.A.L.: Learning Automata: An Introduction. Prentice-Hall, New Jersey (1989)
6. Obaidat, M.S., Papadimitrious, G.I., Pomportsis, A.S.: Learning automata: Theory, paradigms, and applications. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics 32(6), 706–709 (2002)
7. Poznyak, A.S., Najim, K.: Learning Automata and Stochastic Optimization. Springer, Berlin (1997)
8. Thathachar, M.A.L., Sastry, P.S.: Networks of Learning Automata: Techniques for Online Stochastic Optimization. Kluwer Academic, Boston (2003)
9. Tsetlin, M.L.: Automaton Theory and the Modeling of Biological Systems. Academic Press, New York (1973)
10. Thathachar, M.A.L., Sastry, P.S.: Varieties of learning automata: An overview. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics 32(6), 711–722 (2002)
11. Oommen, B.J., 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)
12. Oommen, B.J., Hashem, M.K.: Modeling a Student's Behavior in a Tutorial-*like* System Using Learning Automata. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics SMC-40(B), 481–492 (2010)
13. Oommen, B.J., Hashem, M.K.: Modeling a Domain in a Tutorial-*like* System Using Learning Automata. Acta Cybernetica 19, 635–653 (2010)
14. Oommen, B.J., Hashem, M.K.: Modeling a Student-Classroom Interaction in a Tutorial-*like* System Using Learning Automata. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics SMC-40(B), pp. 29–42 (2010)
15. Thathachar, M.A.L., Oommen, B.J.: Discretized reward-inaction learning automata. Journal of Cybernetics and Information Science 24–29 (Spring 1979)
16. Misra, S., 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)
17. Obaidat, M.S., Papadimitriou, G.I., Pomportsis, A.S., 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)
18. Oommen, B.J., Roberts, T.D.: Continuous learning automata solutions to the capacity assignment problem. IEEE Transactions on Computers C-49, 608–620 (2000)
19. Papadimitriou, G.I., Pomportsis, A.S.: Learning-automata-based TDMA protocols for broadcast communication systems with bursty traffic. IEEE Communication Letters, 107–109 (2000)

20. Atlassis, A.F., Loukas, N.H., Vasilakos, A.V.: The use of learning algorithms in atm networks call admission control problem: A methodology. Computer Networks 34, 341–353 (2000)
21. Atlassis, A.F., Vasilakos, A.V.: The use of reinforcement learning algorithms in traffic control of high speed networks. Advances in Computational Intelligence and Learning, 353–369 (2002)
22. Vasilakos, A., Saltouros, M.P., Atlassis, A.F., 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 (2003)
23. Seredynski, F.: Distributed scheduling using simple learning machines. European Journal of Operational Research 107, 401–413 (1998)
24. Kabudian, J., Meybodi, M.R., 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 2004, Las Vegas, Nevada, pp. 638–642 (2004)
25. Meybodi, M.R., Beigy, H.: New learning automata based algorithms for adaptation of backpropagation algorithm pararmeters. International Journal of Neural Systems 12, 45–67 (2002)
26. Unsal, C., Kachroo, P., 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)
27. Oommen, B.J., Croix, E.D.S.: Graph partitioning using learning automata. IEEE Transactions on Computers C-45, 195–208 (1995)
28. Collins, J.J., Chow, C.C., Imhoff, T.T.: Aperiodic stochastic resonance in excitable systems. Physical Review E 52, R3321–R3324 (1995)
29. Cook, R.L.: Stochastic sampling in computer graphics. ACM Trans. Graph. 5, 51–72 (1986)
30. Barzohar, M., 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)
31. Bertsimas, D.J., Ryzin, G.V.: Stochastic and Dynamic vehicle routing in the euclidean plane with multiple capacitated vehicles. Operations Research 41, 60–76 (1993)
32. Brandeau, M.L., Chiu, S.S.: An overview of representative problems in Location Research. Management Science 35, 645–674 (1989)
33. Bettstetter, C., Hartenstein, H., Pérez-Costa, X.: Stochastic properties of the random waypoint mobility model. Journal Wireless Networks 10, 555–567 (2004)
34. Rowlingson, B.S., Diggle, P.J.: SPLANCS: spatial point pattern analysis code in S-Plus. University of Lancaster, North West Regional Research Laboratory (1991)
35. Paola, M.: Digital simulation of wind field velocity. Journal of Wind Engineering and Industrial Aerodynamics 74-76, 91–109 (1998)
36. Cusumano, J.P., 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)
37. Baddeley, A., Turner, R.: Spatstat: An R package for analyzing spatial point patterns. Journal of Statistical Software 12, 1–42 (2005)
38. Vasilakos, A.V., Saltouros, M.P., Atlassis, A.F., 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)

39. Omar, N., Leite, A.S.: The learning process mediated by intelligent tutoring systems and conceptual learning. In: International Conference On Engineering Education, Rio de Janeiro, p. 20 (1998)
40. Fischetti, E., Gisolfi, A.: From computer-aided instruction to intelligent tutoring systems. Educational Technology 30(8), 7–17 (1990)
41. Winkels, R., Breuker, J.: What's in an ITS? a functional decomposition. In: Costa, E. (ed.) New Directions for Intelligent Tutoring Systems. Spring, Berlin (1990)
42. Self, J.: The defining characteristics of intelligent tutoring systems research: ITSs care, precisely. International Journal of AI in Education 10, 350–364 (1999)
43. Sanders, J.R.: Presented at the 24th annu. meeting joint committee stand. educ. eval. (October 1998), `http://www.jcsee.org/wp-content/uploads/2009/08/JCMinutes98.PDF`