LEARNING AUTOMATA BASED INTELLIGENT TUTORIAL-*LIKE* SYSTEMS

By

M. Khaled Hashem

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

April 2007

Your file Votre référence
ISBN: 978-0-494-27099-8
Our file Notre référence
ISBN: 978-0-494-27099-8

Dedicated with all my love

to

my mother, father, and wife

Farida, Sayed, and Dina

♡

Acknowledgements

First and foremost, I am not able to find the words to express my gratitude to my advisor, Professor John Oommen, for his continuous support, encouragement and for believing in me. During the last few years, I have been a witness of the boundless generosity of his teaching, research, time and advice, so that I could complete the significant amount of work that this Thesis involved. I was always motivated by his words: "Do not get discouraged". He has been like a father, friend and a source of inspiration to me. From the deepest depths of my heart, I will *always* be grateful to him for all that he did for me.

I would also like to express my gratitude to the Thesis Proposal Committee (listed alphabetically, Professor J. P. Corriveau, Professor N. Holtz, and Professor N. Japkowicz) for their invaluable comments during the examination. Their input and the corresponding effect that it had on the Thesis was phenomenal.

I thank Professor Neal Holtz for his continuous help and support in the initial stages of this research. I am grateful for his guidance and encouragement. I also thank the School of Computer Science at Carleton (and in particular Mrs. Claire Ryan) for all the support, especially in the last year.

I am also thankful for the support of my colleagues at McKinsey&Co. They accommodated my hectic work/academic schedule. A very special thanks to Prasoon Sharma and Satty Bhens for their continuous support, encouragement, and trust.

Last, but by no means least, it gives me immense pleasure to thank my parents and my wife. There is nothing that I can say that would do justice to how much I appreciate their love and support. My parents have provided me with endless love, support, and motivation for study for which I am eternally grateful. Dina's love, support, encouragement, and companionship has transformed my journey through this Thesis into a pleasure. She has my everlasting love!

Abstract

The aim of this Thesis is to study, design, and implement systems that could tutor other sub-systems using techniques that traditional *real-life* Teachers use when they teach *real-life* Students. The research undertaken is a result of merging the fields of Intelligent Tutoring Systems (ITS) and Learning Automata (LA), and leads to a paradigm referred to as Intelligent Tutorial-*like* systems. In our proposed novel approach, *every* component incorporates the fundamental principles of LA. Thus, we model the Student (i.e., the learning mechanism) using an LA, and a Classroom of Students, in which each of them is individually represented by a distinct (and possibly different) LA. We also model the Domain and the Teacher using LA paradigms.

Our research also works within a new philosophical perspective. We relax the constraint that "traditional" Tutorial systems have, namely that of assuming that the Teacher is infallible. Rather, we assume that the Teacher is inherently uncertain of the domain knowledge, and is thus of a stochastic nature. However, although he is not absolutely certain about the material being taught, he is also capable of improving his *own* "teaching skills" even while the operation of the system proceeds. Finally, we also attempt to model a realistic learning framework, where the Students can learn not only from the Teacher, but also from other colleague Students in the Classroom.

Apart from designing the various components of the system, we have also implemented a prototype which exhibits all these properties.

The Thesis demonstrates, within an experimental framework, that our system resolves some related fundamental conceptual questions. It also possesses promising features that can lead to benefits in "traditional" Tutorial systems. We believe that we have taken a few small, but significant and pioneering steps, in this direction.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The study of Tutorial systems is one of the more fascinating areas in Computer Science. It encompasses almost every subdiscipline in the field. This is probably the reason why it has been explored so extensively in the literature, and also utilized in various application domains. To be more specific, the researcher who intends to design a good Tutorial system must first and foremost have a thorough knowledge of the application domain which he[1] wants to "teach". He must also have a fair understanding of the cognitive issues at stake during the processes of teaching and learning. This also involves the relevant psychological factors that must be given importance. The designer must also consider the man-machine interface aspects so as to render the whole teaching/learning process user-friendly. This is an area of research in its own right.

The next question involves that of testing how the Student has learned. Additionally, the concept of introducing intelligence into such a Tutorial system is also fascinating. Indeed, this involves many aspects of Artificial Intelligence (AI), and can influence the way the teaching material is presented, the rate at which it is presented, and the feedback that is given to the Student. Tutorial systems that incorporate AI

[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.

techniques are generally referred to as *Intelligent Tutorial Systems.*

Tutorial systems have been developed for numerous application domains. Some of the earliest ones included the teaching of Arithmetic [157, 129, 146], Geography [27], Meteorology [128], Electronics [21], Algebra [125], and the LISP programming [110]. More recently, systems which teach Chess Course [13], Essay Assessment [31], High school Mathematics [59], Physics [68, 149], Algebra word problems [50], Elementary Geometry [107], and Electrical Engineering [160] have also been reported.

Although the field has matured considerably, there are vast avenues open for extensive research. The main consideration that we are concerned about is to see how we could replace a *real-life* Student by an "Entity" - which could be, for example, a component in a cybernetic system, a program, or for that matter any subsystem which needs to learn while it makes its decisions. The intention is that the Tutorial system should be designed to impart knowledge to this "Entity" while working in an interactive manner.

Besides dealing with an artificial Entity, we would also like to open the door to relaxing the constraint that traditional Tutorial systems require, namely the infallibility of the Teacher. The fact is that traditional Tutorial systems assume that the Teacher is perfect – which is a valid assumption as long as one is only dealing with statements of the "truism" type *"2 + 2 = 4"*. However, if the Teacher is, in fact, an Environment or Medium in which the Student is learning, it is conceivable that the decisions and the advice he provides are not always exact. In other words, we would like to relax this infallibility constraint and permit the Teacher himself to be stochastic. This is also one of the philosophical goals of our Thesis.

A system which permits the processes of teaching and learning within such a generalized framework will be referred to as a Tutorial-*like* system.

Completely unrelated to the field of Tutorial systems is an area of research referred to as Learning Automata (LA). These are stochastic machines which are either of a fixed or variable structure, and which are capable of learning in random environments.

The study of the field of LA was initiated in 1962 by the Russian pioneer Tsetlin [144], and the area has virtually exploded in the last decades.

The fundamental goal of this Thesis is to design and implement Tutorial-*like* systems using the theory and fundamentals of LA.

1.1 Motivation and Objectives of the Thesis

The latest developments in the field of Intelligent Tutorial systems seem to indicate that research in the area has stagnated in academia [118]. Producing intelligent tutoring systems that prove to be powerful is important in order to break through the traditional educational barriers by allowing the tailoring of applications to specific user needs and requirements. This need is further augmented with the fact that the Student need not be a *real-life* student but an Entity (as described above), and the Teacher could also be an artificial system. Thus, we believe that designing and implementing Tutorial-*like* systems which possess the features we alluded to, will be a significant step in this research area.

The aim of this research endeavor is to design a Tutorial-*like* system in which *every* component utilizes the fundamental principles of LA. Indeed, we intend to model the Student (i.e., the learning mechanism) using an LA. We also propose to model a *Classroom* of Students - all of them being appropriately represented by *different types* of LA. This, of course, broadens the horizons of both Tutorial systems and the fields of LA because we permit Students to not only learn from the Teacher, but to also learn by interacting with each other.

Not only is this a conceptual goal; we also intend to design and implement a system which can achieve this, albeit within an experimental framework. This, of course, opens avenues for even more fascinating problems such as:

- How does the imperfect Teacher in such a Tutorial-*like* system present his knowledge.
- How is the Domain knowledge represented.
- More importantly, since the Teacher himself is a component of the system, can we incorporate learning capabilities in the Teacher also, thus permitting him to improve his teaching capabilities as the the Student-Teacher interaction progresses.

This Thesis argues that LA-based Intelligent Tutorial-*like* systems can be an easy and useful way to model and implement Tutorial systems, within the expanded horizon mentioned above. We assume that the "Teacher" has an imprecise knowledge of the material to be imparted. While such a model for the Teacher has been studied extensively within the domains of LA, Neural Networks, and reinforcement learning [49, 84, 131], it is quite new to the field of Tutorial Systems. Thus, we believe that *after* our problem has been satisfactorily solved within a machine learning perspective, it can be, hopefully, ported to the application domain of Intelligent Tutorial systems.

In short, the goal of this research is to investigate how the field of Intelligent Tutorial systems and learning automata can be merged to produce Tutorial-*like* systems which have capabilities that are unreported in the literature.

Clearly, from what we have discussed above, there is a marked difference between Tutorial systems and Tutorial-*like* systems. These differences will be clarified in Chapter 3.

We do not claim to have solved all the problems associated with such a design. However, we do believe that we have taken a few small but significant and pioneering steps in this direction. The details of these will be described in more detail during the rest of this Chapter.

The intention of the research in this Thesis is *not* to develop a generic system in which the Teacher is uncertain about the teaching material, or a generic model for the

strategy by which he would impart the material and test the Students. Such a system would encounter enormous hurdles related to the psychological concepts of cognition, teaching, learning and intelligence, and also the system development aspects. In the research work presented in this Thesis, we propose that the problem we are studying be couched within the framework of the general machine learning paradigm, and thus we refer to the proposed system as a Tutorial-*like* system. Thus, it is reasonable for us to interchangeably refer to the "Student" as a "Student Simulator", and to the "Teacher" as the "Teacher Simulator", etc.

1.2 Organization and Contribution of the Thesis

In this Section, we will lay out the structure of the Thesis and identify, what we believe are, the main contributions of every aspect of our research.

Chapter 2 provides a review of the literature pertinent to the Thesis. This Chapter also serves to introduce the main concepts upon which this work is based. It will start with an introduction to the field of stochastic LA, and proceed to a brief overview of the results available in Tutorial systems.

1.2.1 The Overview System Architecture

In Chapter 3, we will first introduce our new concept of Tutorial-*like* systems, list the primary features and characteristics that distinguish them from "traditional" Intelligent Tutorial systems. Thereafter, we will present an overview of the proposed Tutoring-*like* system from the perspective of its overall design and a high-level "bird's-eye view" architecture. In this design, we will catalogue the various modules of the Tutoring-*like* system and how they interact with each other, and how the field of stochastic LA is pervasive within the whole system. Thus, the Chapter, essentially, presents an overall global view of our paradigm. Brief descriptions of each of the modules of the system are then provided so that the reader can recognize the role of

every component in the system and their mutual interaction.

The initial results of this Chapter were presented as a Plenary/Keynote talk at SummerSim'06, the *2006 Summer Simulation Multiconference* in Calgary, July/August 2006, and can be found in [100].

1.2.2 Novel Philosophy to Model a Student

In Chapter 4, we present a new philosophy by which we can model the behavior of a Student in a Tutorial-*like* system using LA. The salient features of this model can be summarized as follows:

- The model of the Student can be inferred using a higher-level of LA, referred to *Meta-LA*.
- The *Meta-LA* attempt to characterize the learning model of the Students (or Student Simulators), even *while* the latter use the Tutorial-*like* system.
- The *Meta-LA*, in turn, use LA as a learning mechanism to try to determine if the Student in question is a fast, normal, or slow learner.

The intention of incorporating such a Student modeling phase is to enable the system to understand how the Student perceives and processes knowledge. This will enable the Tutorial-*like* system to be able to *customize* its teaching according to the capabilities and skills of each Student.

It is conceivable that each Student can be represented by other types of learning mechanisms such as Neural Networks, Reinforcement learning models, etc. We believe that generalizing our paradigm to other learning models will not be too difficult. However, to keep the discussion focused, we shall concentrate only on the premise that the Student is modeled by LA. The problem that we foresee in using other learning models is that of understanding how the Students will impart the knowledge

to their colleagues. In the case of LA, this can be achieved in a straight forward matter by a simple message passing mechanism in which the action probability vector is communicated. If each learning model is a Neural Network, the information to be communicated will probably have to include the number of neurons and their inter-connecting weights, rendering the problem to be even more complex. Thus, for example, the problem would be almost intractable if the network used by each Student is different – i.e., if one Student is modeled using a Kohonen network [60] while the partner that it is communicating with uses a back-propagation algorithm [113]. These issues are enormously complex and we believe that it is outside the scope of our Thesis. In all brevity, we shall restrict ourselves to the LA model of learning.

The work done in this Chapter will be published in the *Proceedings of IEA/AIE 2007: The 20th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems*, Kyoto, Japan, in June, 2007 [46].

1.2.3 Modeling a Student-Classroom Interaction

The field of LA and stochastic learning traditionally deals with the model in which the Teacher (environment) attempts to teach a Student (LA). Baba [9] and others generalized the model when the Student attempts to learn from a multi-Teacher setting. Strictly speaking, this is really a problem in the area of fusion [103], where the Student has to make an intelligent decision based on the response he is getting from a collection of Teachers, and where the responses are sometimes contradictory. As far as we know, the converse problem has not been tackled. This is, indeed, the problem in which a single Teacher is allowed to teach multiple *communicating* Students of varying calibers.

In Chapter 5, rather than invoking the traditional approach of learning in the LA paradigm (in which the Student learns from a Teacher or many Teachers), we sail uncharted waters by departing from this paradigm using principles that, as far as we know, are unreported. These include:

- Allowing the Student to be a member of a *Classroom* of Students.
- Permitting each member of the Classroom to not only learn from the Teacher(s) but to also "extract" information from any of his colleague Students.
- Providing Students with the freedom to choose to work independently or by communicating with his colleagues.
- Permitting Students the ability to accept/discard information received from his colleagues.

In summary, within the context of LA, this Chapter deals with the issues concerning the modeling, decision making process, and testing of such a Student-Classroom interaction philosophy.

1.2.4 Modeling an Increasingly Complex Domain

The question of how to model the Domain in a Tutorial-*like* system is then discussed in Chapter 6. The Chapter will present a novel approach to achieve this for increasingly complex domains as applicable to our Tutorial-*like* system. The salient features of this approach are:

- 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.
- The Tutorial-*like* system is able to present the teaching material in different *chapters*, where each chapter represents a level of difficulty that is harder than the previous one.

To be consistent with our goal, the Domain model is designed within the LA paradigm. We also mention that, to the best of our knowledge, such a modeling of the Environment for increasingly complex problems, is novel to the field of LA.

1.2.5 Modeling Teachers who *Provide* Hints

In Chapter 7, we present a novel approach to model the behavior of a *Teacher* who can provide *hints* within the Tutorial-*like* system. In this model:

- The Teacher is capable of presenting teaching material (using the above mentioned Socratic-type Domain model) *via* multiple-choice questions.
- Since this knowledge is stored in the Domain model in chapters with different levels of complexity, we impart to the Teacher the capability of presenting, to the Students, learning material of varying degrees of difficulty.
- We also impart to the Teacher the ability to assist Students in these more-difficult chapters. This is achieved by him providing them with *hints* that are related to the difficulty of the learning material presented.

The Chapter will demonstrate that, with the help of the *hints* from the Teacher, the Students are able to cope with the process of handling more complex knowledge, and to learn it appropriately. Similar results are unreported in the field of LA.

1.2.6 Modeling the "Learning Process" of a Teacher

The Teacher, in our Tutorial-*like* system, while teaching the Domain knowledge to the Students, is able to "learn" and improve his "teaching skills" while being himself an integral component of the system.

In Chapter 8, we present a model by which the Teacher can accomplish this. As a result, the Teacher will be able to, appropriately, customize his teaching to each Student according to the latter's skill and improvement. The salient features of this model are as follows:

- The Teacher will make use of the Student model provided by the system. We assume that this model is provided by the *Meta-LA* (see Section 1.2.2), and infers the learning capabilities of the Student.
- The Teacher will use a higher-level LA, referred to as the *Meta-Teacher*, which will infer the required customization that the Teacher needs for the particular Student.
- Based on the knowledge inferred from the *Meta-Teacher* and the Student model, the Teacher will be able to customize the teaching material presented, and provide the appropriate *hints* to each individual Student.

The Chapter demonstrates the *adaptability* of the Teacher to the particular needs and skills of each Student.

1.2.7 New Class of Tutorial Systems

In Chapter 9, we present LAB-ITS, the prototype implementation of our LA-Based Intelligent Tutorial-*like* System, and how a researcher can use it. In this prototype, we simulate how Students learn and interact with the Teacher and with each other. The simulation is capable of testing any reasonable hypotheses for the learning of the Students within this framework. Typical hypotheses can be:

- How can we compare the learning of 2 groups, where the first group has 8 colleagues all being Normal learners, and the second group has 5 Superior Students and 3 Weak Students.

- Is a weak learner better-off learning with another set of (a) 8 colleagues (3 Fast, 3 Normal, and 2 Slow learners), where he completely transfers knowledge from them, or (b) 4 Fast colleagues, where he partially utilizes their knowledge.

The Chapter will also present the system's User Interface, and a sequence of screenshots, explaining how the researcher will interact with it. The intention is that the system, in its entirety, will demonstrate all the aspects described in Chapter 3 to Chapter 8.

A typical instantiation of the Tutorial-*like* system would involve:

- Providing the configuration for the Students and the Classroom. For each Student these configurational aspects are needed so that the Student Simulator can function within the Teacher-Student-Classroom loop. The details are:
  - The identity of the Student.
  - The type of Student – being either Fast, Normal, or Below-Normal.
  - The parameters characterizing the Student, i.e., his rate of learning, etc.
  - The intended interaction strategy.
- Providing the configuration of the Domain knowledge.
- Providing the characteristics of the learning Environment, including the way the complexity increases with the chapters.
- Defining the characteristics of the Teacher, and how he will use the *hints* to assist the Students.
- Defining how the Teacher will improve his own teaching abilities.

The simulation of the Tutorial-*like* system can be done either for a single run, or for an ensemble of experiments. At the end of the simulations, the researcher will be able to graphically observe the progress of each Student. These graphs will also show how each Student learned, and the rate of the learning for each Student.

1.3 Summary and Novelty of Contributions

This Chapter provided an overview of the Thesis, and a Chapter-by-Chapter record of the respective contributions. To condense all of these assertions, we present a summary of the contents of the Chapters and their respective novel contributions in Table 1.1. A brief survey of the relevance of the novel contributions to the respective fields is itemized below.

- **Overall Proposed Model**: In this regard, we presented a novel design of a Tutorial-*like* system in which:
  - The Teacher is stochastic, and uncertain about the teaching material.
  - The Student is simulated to study the Domain knowledge.
  - The Domain knowledge contains uncertain course material.
  - The Teacher is dealing with a *School* of Students who learn from him and from each other.
- **Modeling the Student's Behavior**: In this regard, we introduced the concept of a *Meta-LA*, as a higher-level automaton, which tries to infer the learning model of the lower-level automata. This can actually be perceived as a new Pattern Recognition classifier. This model presents a novel synchronous interconnected automata structure, for which there are no direct connections between the higher-level LA and the lower-level LA.
- **Modeling Student-Classroom Interaction**: This model presents a novel LA contribution, in which the Students (LA) learn not only from the Teacher (Environment), but are also able to "extract" information and learn from each other. It also presents a novel implementation in Tutorial systems, where Students can communicate between each other, so as to monitor their learning progress. This communication is facilitated by the introduction of the so-called *Tactic-LA*, which simulates the decisions for the Students' interactions.

- **Modeling the Domain Knowledge**: This model represents a novel contribution for the LA field, in which the complexity/difficulty of the Environment can be gradually increased by reducing the range of the penalty probabilities of the corresponding actions. This results in increasing the difficulty for the Students to learn the required knowledge.

- **Modeling the Teacher's Behavior**: This model presents a novel contribution in the field of LA in which the Teacher is assisting the Student to deal with increasingly complex material by providing him with *hints*. The convergence of the Student is influenced by the response from the Teacher and from the *hints* provided by the Teacher.

- **Modeling the "Learning Process" of the Teacher**: This model presents a novel contribution to the field of LA, in which the Teacher uses a higher-level automaton (*Meta-Teacher*) to infer the required customization that he needs for a specific Student. This model, again, represents a novel synchronous interconnected automata structure, in which there are no direct connections between the higher-level LA and the lower-level LA.

- **The Overall Tutorial-*like* System**: The simulator presents a novel implementation of a prototype of a Tutorial-*like* system. In this simulator, a researcher can observe the learning progress of the Student Simulators in different configurations. This can enable him to test any reasonable hypotheses related to how the Student(s) can learn from the Teacher and from each other, depending on their interaction strategy.

Chapter	Title	Content	Novel Contribution
3	Overall Proposed Model	• Tutorial-*like* systems (Overall Model) • Rationale for this Project	No comparable work in Tutorial Systems
4	Modeling the Student's Behavior	• The system infers the Student's learning model using *Meta-LA*	No comparable work in LA
5	Modeling Student-Classroom Interaction	• Models how a Student can learn from colleagues	No comparable work in LA/Tutorial Systems
6	Modeling the Domain Knowledge	• Modeling the Domain knowledge with increasing complexity	No comparable work in LA
7	Modeling the Teacher's Behavior	• Modeling a Teacher, who can provide hints to the Students	No comparable work in LA
8	Modeling the "Learning Process" of the *Teacher*	• Modeling how the Teacher can "learn" and improve his "teaching skills"	No comparable work in LA
9	The Overall Tutorial-*like* System	• All the modules of the Tutorial-*like* system functioning together • Implementation and the UI	No comparable work in Tutorial Systems

Table 1.1: Summary of the Overall Contributions of the Thesis, Presented in a Chapter-by-Chapter Format.

Chapter 2

Survey of State of the Art

2.1 Chapter Overview

In this Chapter, we will present a survey of the state of the art research in the relevant areas pertinent to this Thesis. In the first part, we will introduce Learning Automata (LA) as tool for adaptive learning, and will present their different models, algorithms, and classifications. In the second part, we will present Tutorial systems, their models and the relevant state-of-the-art research in *this* field. In our research in this Thesis, LA will be the learning tool used to design novel families of Tutorial-*like* systems.

2.2 Adaptive and Learning Systems

Adaptation and learning are terms used to describe "behavior modification in natural organism as well as machines" [85]. The definition of an adaptive system has diverse viewpoints. Zadeh [159] argued that "the difficulty in defining the notion of adaptivity is due to a lack of a clear differentiation between the external manifestations of adaptive behavior and the internal mechanism by which it is achieved".

Drenick and Shahbender [36] introduced the term "adaptive system" in control theory to represent control systems that monitor their own performance and adjust their parameters in the direction of better performance [80]. Eveleigh [38] defined an adaptive system as a system "which is provided with a means of continuously monitoring its own performance in relation to a given figure of merit or optimal condition and means of modifying its own parameters by a closed-loop action so as to approach this optimum".

Learning can be defined as any "permanent change in behavior" [89] or the "ability to improve performance" as a result of past experience [37]. A learning system should have the ability to improve its performance with time. Thathachar and Sastry [142] defined that a machine is said to learn if "it improves its performance through experience gained over a period of time without complete information about the environment in which it operates". Self-improvement in performance could be regarded as the *hallmark* of learning machines.

2.3 Stochastic Automaton

The Stochastic Automaton represents one approach to machine learning that is useful in a variety of situations involving learning from examples, or learning by doing [142]. The stochastic automaton tries to reach a solution to a problem without *any* information about the optimal action. By interacting with an Environment, a stochastic automaton can be used to learn the optimal action offered by that Environment [84, 98]. A random action is selected, based on a probability vector, and then from the observation of the Environment's response, the action probabilities are updated, and the procedure is repeated. A stochastic automaton that behaves in this way to improve its performance is called a learning automaton (LA) [84, 89]. The learning approach that LA use may be described as *learning by doing* [142].

Research in LA started with Tsetlin [144] who introduced the use of deterministic and stochastic automata operating in a random Environment as a learning model.

He proved that they can be asymptotically optimal under some conditions [84]. We explain this later in this Chapter.

The term "Learning Automata" was first publicized in the survey paper by Narendra and Thathachar [82]. The goal of LA is to "determine the optimal action out of a set of allowable actions" [3]. 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 [141].

2.3.1 Learning Automaton Classifications

Learning automata can be classified in three categories: deterministic, fixed structure stochastic, and variable structure stochastic [79]. For deterministic automata, the transition and the output matrices are deterministic [150]. For fixed structure stochastic automata (FSSA), their transitions are determined by state transition probabilities that are fixed with time [84].

Variable structure stochastic automata (VSSA) have a stochastic transition matrix whose transition probabilities or action probabilities are adjusted as the learning system operates [84]. Varshavskii and Vorontsova [150] were the first to introduce the variable structure automata that updated transition probabilities. Fu and McMurtry [42] then introduced the variable structure automata that updated action probabilities. The use of stochastic automata leads to a reduction of states in comparison with deterministic automata [150].

2.3.2 Basic Mathematical Definition

Models in which the output of the Environment can take only one of two values (0 and 1 for example) are referred to as P-models. In S-models, the output can take

the form of a continuous random variable in the interval [0, 1]. In Q-models, the Environment output can have discrete values in the interval [0, 1], for example {0, 0.1, 0.2,...} or {0, 0.05, 0.10, 0.15,.....}.

The following is the mathematical definition of the LA model (FSSA in the P-model Environment). This includes the definition of the automaton, the interacting Environment, the objectives of this interaction, and the learning method [37]:

An Environment is defined by a triple $\{\underline{\alpha}, \underline{\beta}, \underline{c}\}$, where,

$\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots., \alpha_r\}$ is the set of actions (input to the Environment}.

$\underline{\beta} = \{0, 1\}$ is the set of responses (output of the Environment}.

$\underline{c} = \{c_1, c_2, \ldots., c_r\}$ is an unknown penalty probability set, where

$$c_i = \Pr[\beta(t) = 1 \mid \alpha(t) = \alpha_i].$$

The Learning Automaton is defined by a quintuple $\{\underline{\Phi}, \underline{\alpha}, \underline{\beta}, \mathbf{F}(.,.,.), \mathbf{G}(.)\}$, where,

$\underline{\Phi} = \{\phi_1, \phi_2, \ldots, \phi_s\}$ is the set of the internal states.

$\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the set of actions (output of the automaton).

$\underline{\beta} = \{0, 1\}$ is the set of feedback signals (input of the automaton).

$\mathbf{F}(.,.,.)$: $\underline{\Phi} \times \underline{\beta} \rightarrow \underline{\Phi}$ is the state transition mechanism according to which the next state is chosen (depending on the current state and the Environment response).

$\mathbf{G}(.)$: $\underline{\Phi} \rightarrow \underline{\alpha}$ is the action selection mechanism according to which the current action is chosen (depending on the current state).

For VSSA, at each instant t, the automaton, according to the action probability vector $\mathrm{P}(t)$, selects randomly an action $\alpha(t) = \alpha_i$ from the finite action set $\underline{\alpha}$. The

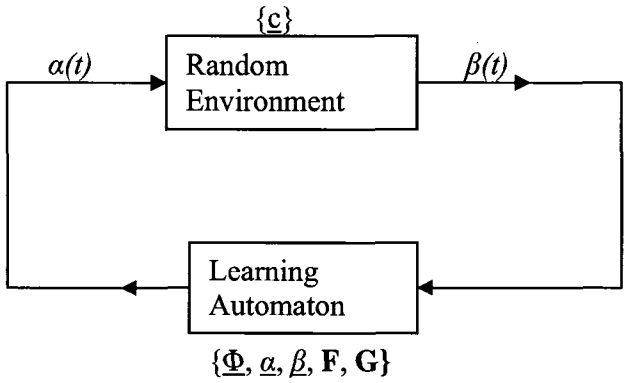

Figure 2.1: Interaction and Feedback Between Automaton and Environment.

probability that the automaton selects action α_i, at time t, is the action probability $p_i(t) = \Pr[\alpha(t) = \alpha_i]$, where:

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

The Environment responds with $\beta(t) = 0$ when the response is favorable (reward) and $\beta(t) = 1$ when the response is unfavorable (penalty). The Environment's response to action α_i is chosen according to the unknown penalty probability $c_i = \Pr[\beta(t) = 1 \mid \alpha(t) = \alpha_i] \; \forall\, i$. The Environment is characterized by the set of penalty probabilities $\underline{c} = \{c_1, c_2, \ldots, c_r\}$. The Environment's reward probability is $d_i = 1 - c_i$, $i = 1, \ldots, r$. The Environment's penalty probabilities $\{c_i\}$ are unknown to the automaton.

The Environment typically has an optimal action, namely the one associated with the minimum penalty probability $c^* = \min_i \{c_i\}$. The performance of the automaton is usually evaluated by the average cost for a given action probability vector:

$$\begin{aligned} M(t) &= E[\beta(t)|p(t)] = \Pr[\beta(t) = 1|p(t)] \\ &= \sum_{i=1}^{r} \Pr[\beta(t) = 1|\alpha(t) = \alpha_i] \Pr[\alpha(t) = \alpha_i] \\ &= \sum_{i=1}^{r} c_i p_i(t). \end{aligned}$$

Learning is achieved when the automaton chooses an action according to the

current action probability vector, and then updates its action probability vector according to the response from the Environment. If the automaton is successful, this will, in general, lead to the selection of the superior actions with a higher probability and the reduction of the cost, $M(t)$.

2.3.3 Stationary and Non-stationary Environments

The Environment represents the system which communicates with the learning system and supplies it with information [79]. Environment types can be classified as being stationary and non-stationary. A stationary Environment is one that does not change with time, as opposed to a non-stationary Environment which changes with time. Thus, in a non-stationary Environment, the penalty probabilities $\{c_i\}$ corresponding to the automaton action varies with time.

2.3.4 Variable Structure Stochastic Automata and Reinforcement Schemes

In VSSA, the automata are represented by the quadruple:

$\{\underline{\alpha}, \underline{\beta}, T, \mathrm{P}\}$

in which $\underline{\alpha}$ is a finite set of actions, $\underline{\beta}$ is a set of outputs from the Environment, T is the updating algorithm, or the reinforcement scheme, and P is the action probability vector such that $\mathrm{P}(t) = \big(p_1(t), p_2(t), \ldots, p_r(t)\big)$ with $p_i(t) = \Pr[\alpha(t) = \alpha_i]$. In general, learning or reinforcement schemes can be represented by either [84]:

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

$f_{ij}^{\beta}(t+1) = T'[\mathrm{F}(t), \phi(t), \beta(t)]$,

where T and T' are the corresponding mappings. In the first equation, at $(t+1)$ the action probability vector is updated on the basis of its previous value, the action

$\alpha(t)$, and the input $\beta(t)$. In the second equation, the transition probability $f_{ij}^{\beta}(t+1)$ relating a transition $\phi_i \rightarrow \phi_j$ at the instant $(t+1)$ for an input β depends on the value of this transition probability matrix at the instant t, the state of the automaton at t, and the actual input $\beta(t)$ at t.

Learning automata can also be classified in terms of their Markovian representation as being either absorbing or ergodic [93]:

- *Absorbing* algorithm: If the mapping T is chosen in a manner that the Markov process has absorbing states, the algorithm is referred to as an *Absorbing* algorithm [3].
- *Ergodic* algorithm: In this case, the LA converges with a distribution that is independent of the initial distribution of the action probabilities [93].

The updating schemes of the learning algorithm can also be classified according to the functional forms of the updating function as:

- *Linear algorithms* are the ones in which P(t+1) is a linear function of P(t).
- *Non-linear algorithms* are the ones in which P(t+1) is a non-linear function of P(t).

Different learning schemes could be used for VSSA. For example, if a chosen action α_i is rewarded, the probability for the current action is increased, and the probabilities for all the other actions are decreased. If the chosen action α_i is penalized, the probability of the current action is decreased, and the probabilities of the other actions are increased. This leads to the following different types of learning schemes [75, 84]:

- Reward-Penalty (RP): The action probability vector is always updated regardless of whether the automaton is rewarded or penalized.

- Inaction-Penalty (IP): The action probability vector is updated when the automaton is penalized. When the automaton is rewarded, the action probability vector is not updated.
- Reward-Inaction (RI): The action probability vector is updated when the automaton is rewarded. When the automaton is penalized, the action probability vector is unchanged.

To clarify issues, we consider a VSSA with r actions in a stationary Environment with $\underline{\beta}=\{0, 1\}$. The general scheme for updating action probabilities is as follows [147]:

If $\alpha(t) = \alpha_i$:

when $\beta = 0$:

$$p_j(t+1) = p_j(t) - \mathrm{g}_i(\mathrm{P}(t)) \quad \text{for all} j \neq i$$

$$p_i(t+1) = p_i(t) + \sum_{\substack{k=1 \\ k \neq i}}^{r} \mathrm{g}_k(\mathrm{P}(t))$$

when $\beta = 1$:

$$p_j(t+1) = p_j(t) + h_i(\mathrm{P}(t)) \quad \text{for all} j \neq i$$

$$p_i(t+1) = p_i(t) - \sum_{\substack{k=1 \\ k \neq i}}^{r} h_k(\mathrm{P}(t))$$

where g_k and h_k (k=1,2,...,r) are continuous, nonnegative functions satisfying the following conditions:

$$0 < \mathrm{g}_k(\mathrm{P}(t)) < p_k(t)$$

$$0 < \sum_{\substack{k=1 \\ k \neq i}}^{r} [p_k(t) + h_k(\mathrm{P}(t))] < 1$$

for all i=1,2,...,r, and all probabilities p_k in the open interval (0, 1). The two constraints on the update functions guarantee that the sum of all action probabilities is unity at every time step.

For linear VSSA, three learning schemes demonstrate the three main learning philosophies: The Linear Reward-Penalty scheme (L_{RP}), the Linear Reward-Inaction scheme (L_{RI}), and the Linear Inaction-Penalty scheme (L_{IP}).

Considering a LA with two actions and applying the previous learning schemes [84], let:

$$g_i(P(t)) = ap_j(t)$$

and

$$h_j(P(t)) = b(1 - p_j(t)).$$

In the above two equations, a and b are reward and penalty parameters, where $0 < a < 1$, $0 \leq b < 1$.

These schemes exhibit significantly different characteristics. We describe below the characteristics and optimality phenomena of the above three schemes. In this interest of simplicity, we present below only their 2-action variants. The corresponding r-action versions are described in [84].

2.3.4.1 Linear Reward-Penalty Scheme (L_{RP})

All L_{RP} schemes are ergodic. The specific case when $a = b$ is called the symmetric L_{RP} scheme. The asymptotic value of the average penalty of the symmetric L_{RP} scheme is given by:

$$\lim_{t\to\infty} E[M(t)] = \frac{2c_1c_2}{c_1+c_2} < \frac{c_1+c_2}{2} \triangleq M_0.$$

for the 2-action system. The middle inequality of the above equation proves that the L_{RP} is expedient for all initial conditions and all stationary Environments [84]. In addition, because the scheme is ergodic, it is suitable for a non-stationary Environment.

However, the L_{RP} scheme is not absolutely expedient nor it is ϵ-optimal.

Choosing an arbitrarily small $b > 0$ (where $0 < b << a$) leads to the Linear Reward-ϵ Penalty scheme ($L_{R-\epsilon P}$). The $L_{R-\epsilon P}$ has the same Markovian behavior, but very different asymptotic characteristics. It is ϵ-optimal in all random Environments [84].

2.3.4.2 Linear Reward-Inaction Scheme (L_{RI})

The L_{RI} can be derived when $b = 0$. It is an absorbing scheme, which makes it inappropriate to be used in non-stationary Environments. It has been proven to be ϵ-optimal in all stationary Environments [84]. If the initial probability is $[0.5, 0.5]^T$, then $p_1(t)$ converges to 0 with a higher probability when $c_1 > c_2$ and to 1 with a higher probability when $c_2 > c_1$. This probability of convergence becomes arbitrarily close to unity as a is made arbitrarily small.

2.3.4.3 Linear Inaction-Penalty Scheme (L_{IP})

The specific case when $a = 0$ and b $\neq 0$ is called the Linear Inaction-Penalty scheme (L_{IP}). It has been proved that this scheme is ergodic and expedient [61]. Although the probability of the optimal action grows higher than others, it *can never* reach unity, and is thus never ϵ-optimal. In fact, it approaches $1/(r\text{-}1)$ [147].

2.3.5 Learning Algorithm Efficiency

Today, in the literature, a LA roughly corresponds to what was called *"variable structure stochastic automaton"* [141]; therefore, the remaining part of this Thesis will equivalently refer to LA as VSSA.

Research in LA is mainly focused on solving three important problems, namely, improving the low speed of learning, increasing the accuracy of convergence, and

improving the tracking ability when interacting with a non-stationary Environment [10]. The design problem is to specify T such that the automaton learns more about the Environment, and thus improves its performance and reduces its average loss $M(t)$.

An automaton is said to be *expedient* if:

$$\lim_{n\to\infty} E[M(t)] < M_0,$$

where M_0 is the loss of a *pure chance* automaton and is defined by:

$$M_0 = (1/r) \sum_{i=1}^{r} c_i.$$

Also, a learning automaton is said to be *absolutely expedient* if:

$$E[M(t+1)|\mathrm{P}(t)] < M(t),$$

where $\mathrm{P}(t)$ is the action probability vector.

Absolute expedient schemes such as the L_{RI} scheme interact ideally with stationary Environments [84]. However, they are not useful with non-stationary Environments as the automaton tends to get trapped in an absorbing state. Ergodic schemes such as the L_{RP} and $L_{R-\epsilon P}$ are used in these Environments. Necessary and sufficient conditions for an LA to be absolutely expedient are derived in [62, 84].

In practice, accuracy and speed are important quantities for an automaton; it is desirable to have maximum accuracy and the best possible speed. However, as is well known in Systems Theory, the two requirements frequently conflict [84]. Also, the behavior of the automaton tends to slow when the number of actions increases [79].

In the same spirit, a learning automata is said to be *optimal* if:

$$lim_{n\to\infty} E[M(t)] = c^*.$$

An optimal scheme would, without explicitly computing the estimates of the penalty probability, choose w.p. 1, the action corresponding to the minimum penalty

probability [84]. Optimality is desired in *stationary* Environments, but a sub-optimal performance may be preferred in *non-stationary* Environments. In this kind of Environment, the automata must not only learn the characteristics of the Environment, but also forget old characteristics and acquire new ones in response to the time-varying situation. In non-stationary Environments an automaton could get locked to an action which was originally optimal but later become sub-optimal. It could also respond to the Environment changes but not sufficiently quickly because it is too heavily committed to a previously optimal action.

In practice, ϵ-optimal[1] automata are the closest realization of optimal LA. An automata is said to be ϵ-optimal if:

$$\lim_{t\to\infty} E[M(t)] < c^* + \epsilon,$$

where $\epsilon > 0$ can be made arbitrary small by controlling some parameters of the LA.

Informally, an automaton is said to be ϵ-optimal if "given enough time and given a large enough internal parameter, the probability of picking the best action almost all of the time can be made as close to unity as desired" [64].

2.3.6 Discretized Learning Automata

The original work on LA focused on continuous schemes, in which the action probabilities could take any real value in the interval [0, 1]. Thathachar and Oommen [134] first suggested that LA could be improved if the probability space could be rendered discrete. They introduced the concept of discretized LA by restricting the probability of choosing only from a finite set of values in the closed interval [0, 1]. Therefore, the action probability would be updated in steps instead of being updated in a continuous manner.

Discretized VSSA can be viewed as a hybrid of FSSA and VSSA [64]. They are

[1]It has also been referred to as asymptotically optimal in earlier literature.

similar to FSSA because they also consist of a finite set of "states". On the other hand, they are VSSA because they have an action probability vector that constantly evolves with time. Discrete algorithms can be classified into two categories, linear and non-linear. They are linear if the probability values are uniformly spaced in the interval of [0, 1]. Non-linear schemes have probability values that are not uniformly spaced in the interval [0, 1] [64].

Lanctôt and Oommen defined three advantages of using discretized algorithms. The original motivation of using a discretized algorithm was to increase the speed of convergence and to eliminate the assumption that the random number generator could generate real numbers with arbitrary precision. The probability value can converge to unity in steps rather than approach the value unity asymptotically.

Another benefit of using a discretized algorithm is that it eliminates the stringency imposed on the random number generators. While, in theory, the random number generator is assumed to yield any number between [0, 1], however, in practice, because all machine implementations uses pseudo-RNGs, only a finite number of these values are available. A third advantage is that because the discretized algorithm uses add and subtract operations rather than multiply operations, it consumes less memory and executes faster than the continuous LA.

Oommen and his co-authors [93, 95, 99, 98] have proved that various discretized automata can be ϵ-optimal in all Environments. Discretized learning automata have been proven to converge faster than their continuous counterparts [3, 37, 95, 98].

To be more specific, Oommen and Christensen [95] proved that the DL_{RP} scheme is ergodic and ϵ-optimal in all random Environments where $c_{min} < 0.5$. They also introduced the Modified Discretized Linear Reward-Penalty MDL_{RP} scheme, in which the Environment responses are filtered before being fed back to the automaton. The MDL_{RP} was proven to be ergodic and ϵ-optimal in all random Environments. Oommen and Christensen also created an absorbing version of the DL_{RP} by adding an artificial absorbing barrier to the scheme, denoted by ADL_{RP}. That scheme was proven to be ϵ-optimal in all random Environments. The ADL_{RP} scheme is the

Algorithm	Type	Markov Chain Characterization	Convergence Behavior
L_{RI}	Continuous	Absorbing	ϵ-optimal in all Environments
DL_{RI}	Discrete	Absorbing	ϵ-optimal in all Environments
L_{IP}	Continuous	Ergodic	Expedient
DL_{IP}	Discrete	Ergodic	Expedient
ADL_{IP}	Discrete	Absorbing	ϵ-optimal in all Environments
L_{RP}	Continuous	Ergodic	Expedient in all stationary Environments
DL_{RP}	Discrete	Ergodic	ϵ-optimal if $c_{min} < 0.5$
ADL_{RP}	Discrete	Absorbing	ϵ-optimal in all Environments
MDL_{RP}	Discrete	Ergodic	ϵ-optimal in all Environments

Table 2.1: Comparison between Continuous and Discrete Linear VSSA [1].

only known symmetric linear reward-penalty scheme that is ϵ-optimal in all random Environments.

The analogous properties of the Discretized Linear Reward-Inaction scheme (DL_{RI}), and the Discretized Linear Inaction-Penalty scheme (DL_{IP}) can be found in [93, 99, 134]. Table 2.1 shows the different asymptotic properties and convergence characteristics of all the linear continuous and discrete VSSA.

2.3.7 The Estimator Algorithms

Another new class of algorithms capable of improving the rate of convergence of the LA was introduced by Thathachar and Sastry by proposing the so called Estimator Algorithms [116, 137]. In their novel approach, 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 *estimate* vector. While in non-estimator algorithms the probability vector is updated based on the Environment's response, in an estimator algorithm the update is based on *both* the Environment's response and the estimate vector. Thus, it is easy to observe cases where an action

is rewarded while the probability of choosing *another* action is increased.

While Thathachar and Sastry introduced the continuous version of the estimator algorithm where the probability of choosing an action can be any real number in the closed interval [0,1], Oommen *et al.* introduced various discretized versions of the algorithms [63, 101]. These discretized versions enhance both speed and accuracy of the convergence.

2.3.8 The Pursuit Algorithm

The Pursuit Algorithm is a class of estimator algorithms., and is characterized by the fact that it pursues what it reckons to be the optimal action [138]. The Pursuit Algorithm involves three steps [138]. The first step is to pick the action $\alpha(t)$ based on the probability distribution $\mathrm{P}(t)$. In the second step, if the automaton is rewarded, the component of $\mathrm{P}(t)$, whose current estimate of reward is maximal (the current optimal action), is increased, while all other actions probabilities are decreased. Finally, the third step is to update the running estimates of the probability of being rewarded. For calculating the estimates, two more vectors are introduced: $\mathrm{W}(t)$ and $\mathrm{Z}(t)$, where z_i is the number of times the i^{th} action has been chosen up to time t; and w_i is the number of times the i^{th} action has been rewarded up to time t. The estimate vector $\hat{D}(t)$ can be computed using the following formula:

$$\hat{d}_i(t) = \frac{w_i(t)}{z_i(t)} \qquad \forall i = 1, 2, \ldots, r.$$

The continuous reward-penalty version of the Pursuit Algorithm can be found in [1, 64].

The CP_{RP} algorithm is similar in design to the L_{RP} in the sense that, in the case of a reward or penalty, they modify the action probability vector $\mathrm{P}(t)$. They differ in the way they approach the solution. The L_{RP} moves the $\mathrm{P}(t)$ in the direction of the most recently rewarded, or not penalized, action. On the other hand, the CP_{RP} moves $\mathrm{P}(t)$ in the direction of the action which has the highest reward estimate.

Thathachar and Sastry [137] proved that the Pursuit Algorithm is ϵ-optimal in all stationary Environments. The Pursuit Algorithm was five to seven times faster than the L_{RI} algorithm [64].

2.3.9 The TS Estimator Algorithm

Thathachar and Sastry in [136] introduced a more complex estimator algorithm. It was referred to in the literature as the TS Estimator (TSE) algorithm [64]. The algorithm increases the probabilities for all the actions that have higher estimates than the estimate of the chosen action, and decreases the probabilities of all the actions with smaller estimates. As an estimator algorithm, the probabilities are updated based on both the reward estimates $\hat{\mathrm{D}}(t)$ and the action probability vector $\mathrm{P}(t)$.

The algorithm treats probability components with an estimate of reward greater than $\hat{d}_i$ differently from those with a value lower than $\hat{d}_i$. To achieve this, Thathachar and Sastry introduced an indicator function $\mathrm{S_{ij}}$ which is zero if $\hat{d}_j$ is bigger than $\hat{d}_i$, otherwise it is unity. The details of the TSE Algorithm can be found in [1, 64, 75], and the primary differences between the TSE Algorithm and the Pursuit Algorithm are given in [1].

The TSE algorithm was proven to be ϵ-optimal [136]. Also, simulation comparisons with the $\mathrm{L_{RI}}$ scheme were done by Thathachar and Sastry. They have shown that, for the same accuracy, the TSE Algorithm would converge at least seven times faster than the L_{RI} scheme [136].

2.3.10 Generalized Pursuit Algorithm

Agache and Oommen [2, 3] introduced a general version of the Pursuit Algorithm introduced by Thathachar and Sastry in [138]. In their algorithm, the *Generalized*

Pursuit Algorithm, Agache and Oommen generalized the traditional Pursuit Algorithm by pursuing all the actions with higher reward estimates than the chosen action, thus minimizing the probability of pursuing a wrong action. The performance of the new algorithm was compared quantitatively with existing pursuit algorithms and it was proven to be the fastest continuous converging scheme. It was also proven to be ϵ-optimal in all stationary Environments. The formal algorithm can be found in [3].

2.3.11 Discrete Estimator Algorithms

Lanctôt and Oommen [64] introduced a new class of LA, which is called Discrete Estimator Algorithms (DEA). In this class, the estimator algorithm is discretized, where the probability vector are allowed to take a finite set of discrete values in the closed interval [0,1] and estimate vectors are used to update the probability vector in order to utilize the benefits of estimator algorithms.

Lanctôt and Oommen [64] presented two sufficient conditions required for any member of the DEA family so that it would be ϵ-optimal. These two properties are the *Property of Moderation* and the *Monotone Property.*

- Property of Moderation: This property states that a DEA must implicitly specify an upper bound on the amount any action probability can decrease during a single iteration. If a DEA has r actions and a resolution parameter n, in order to satisfy this property, the maximum magnitude by which an action probability can decrease per iteration is bounded by $1/rn$.
- Monotone Property: If α_m, which is the estimate of reward of an action, remains the maximum estimate subsequent to a certain point of time, then, a DEA possesses the Monotone Property if it steadily increases the probability of choosing α_m to unity.

2.3.11.1 Discrete Pursuit Algorithm

The Discrete Pursuit Algorithm (DPA) follows the same strategy followed by the continuous Pursuit Algorithm, except that the probability changes are made in discrete steps. Therefore, the equations in the continuous Pursuit Algorithm that involve multiplication by the learning parameter λ are substituted by the addition or subtraction of the smallest step size.

The DPA works like the continuous Pursuit Algorithm, except that the components of the probability vector are increased by integral multiples of the smallest step size Δ, where $\Delta = 1/\mathrm{rn}$. In the case of rewarding the automaton when it has not converged, all the non-zero action probabilities are decreased by Δ except the one with the highest estimate of the reward probability. To keep the sum of probability vector components to be unity, this action probability is increased by the appropriate amount. The DPA algorithm can be found in [64], where Lanctôt and Oommen proved that the scheme is ϵ-optimal by showing that the DPA algorithm possessed both the properties of moderation and monotonicity. They experimentally demonstrated that the DPA is at least 60% faster than the C_{PR} algorithm.

2.3.11.2 The Discrete TSE Algorithm

Lanctôt and Oommen [64] discretized the TSE Algorithm, in what they called the Discrete TSE (DTSE) Algorithm. Similar to the continuous TSE Algorithm, the DTSE is a far complex scheme.

Again, Lanctôt and Oommen [64] have proven that the DTSE Algorithm is ϵ-optimal. This optimality is demonstrated by proving that the DTSE Algorithm has the two necessary properties, moderation and monotonicity, while also preserving as many qualities of the continuous TSE Algorithm as possible.

The DTSE Algorithm was shown to be a two parameter system. While in the DPA, the smallest step size is $\Delta = 1/\mathrm{rn}$, in the DTSE Algorithm it is $\Delta = 1/\mathrm{rn}\theta$,

where θ represents the largest integer multiple of Δ that any single component of the probability vector can decrease by in a single iteration.

2.3.11.3 Discretized Generalized Pursuit Algorithm

Agache and Oommen [3] discretized the Generalized Pursuit Algorithm, in what they called the Discretized Generalized Pursuit Algorithm (DGPA), which generalizes the concepts of the Pursuit Algorithm by "pursuing" all the actions that have higher estimates than the current chosen action. This algorithm is unique in the family of discretized LA. While the typical discretized schemes moves the probability vector in discrete steps that are predefined in advance, the DGPA moves the probability vector in discrete unequal steps.

The DGPA, at each iteration, counts how many actions have higher estimates than the current chosen action, denoted by K(t). The DGPA algorithm increases the probability of all the actions with higher estimates with the amount Δ/K(t), and decreases the probabilities for all the other actions with the amount Δ/(r-K(t)), where $\Delta = 1/rn$ denotes the resolution step, and n the resolution parameter. The DGPA algorithm has been proven to be ϵ-optimal because it possesses the moderation and monotone properties [3].

2.3.12 Rate of Convergence for Discretized Pursuit Algorithms

The speed of convergence of the different continuous and the discretized versions of the estimator algorithms have been experminetly evaluated by Oommen and his co-authors [3, 64]. In [64], a comparison of the rate of convergence for the continuous and discretized Pursuit and the TSE Algorithms have been performed. The discretized Pursuit Algorithm was shown to be at least 60% faster than the continuous Pursuit Algorithm. For the TSE Algorithm, the discretized TSE Algorithm was 50%-70%

faster than the continuous TSE Algorithm. Table 2.2 reports the number of iterations that was required to attain the required convergence criteria in two benchmark Environments, E_A and E_B.

Environment	Algorithm	Continuous	Discrete
$\mathbf{E_A}$	Pursuit	1140	799
	TSE	310	207
$\mathbf{E_B}$	Pursuit	2570	1770
	TSE	583	563

Reward probabilities are:										
$\mathbf{E_A}$:	0.7	0.5	0.3	0.2	0.4	0.5	0.4	0.3	0.5	0.2
$\mathbf{E_B}$:	0.1	0.45	0.84	0.76	0.2	0.4	0.6	0.7	0.5	0.3

Table 2.2: Comparison of the Discrete and Continuous Estimator Algorithms in Benchmark 10-Action Environments [64].

Agache and Oommen [3] also compared the convergence rate of the GPA and DPGA for the benchmark ten-action Environments. The results also proved that the DGPA does converge faster than the GPA (results are shown in Table 2.3). In fact, Agache and Oommen have proved that the GPA algorithm is the fastest continuous pursuit algorithm, and that the DGPA was the fastest converging discretized pursuit estimator algorithm of its era.

Papadimitriou *et al.* [105] introduced a new class of learning algorithm, which is the stochastic estimator (SE) LA. They proved that it is ϵ-optimal in every stationary Environment. They also, experimentally, claimed that in stationary P-model Environments, the SE_{RI} converges faster than the DGPA algorithm, achieving a 52% increase in the speed of convergence in comparison to DGPA.

Environment	GPA		DGPA	
	λ	No. Of Iterations	N	No. Of Iterations
$\mathbf{E_A}$	0.0127	948.03	24	633.64
$\mathbf{E_B}$	0.0041	2759.02	52	1307.76

Reward probabilities are:										
$\mathbf{E_A}$:	0.7	0.5	0.3	0.2	0.4	0.5	0.4	0.3	0.5	0.2
$\mathbf{E_B}$:	0.1	0.45	0.84	0.76	0.2	0.4	0.6	0.7	0.5	0.3

Table 2.3: Performance of the GPA in Benchmark 10-Action Environments for which Exact Convergence was Required in 750 Experiments.

2.3.13 LA Structure Classification

Thathachar and Sastry [141] classified LA according to its structure into four groups; finite action-set learning automata, parameterized learning automata, general learning automata, and continuous action-set learning automata.

2.3.13.1 Finite Action-set Learning Automaton (FALA)

This covers most of the algorithms presented in previous sections above and will be the primary focus of this Thesis. This represents the original notion of LA in which the action set is always considered to be finite. Most of the algorithms in this category assure convergence to a local maximizer. Many algorithms, such as the L_{RI} and the estimator algorithms such as the Pursuit Algorithm, have been proved to be ϵ-optimal. As explained in the previous sections, Oommen and his co-authors [93, 95, 99] introduced a new class of FALA, the discretized LA, in which the action probability can take values from a finite set, and they proved that various discretized automata can be ϵ-optimal in all Environments. It has been shown that, in general, discretized LA converge faster than their continuous counterparts [37, 95, 99].

Other algorithms that fall under FALA are the *estimator* family of algorithms, introduced by Thathachar and Sastry [138], that use the estimates of the reward responses from the Environment to update the action probability vector. This includes the continuous *Pursuit* Algorithm CP_{RP}, the TSE Algorithm and its counterpart discretized algorithm, the Discretized Pursuit Algorithm DP_{RP}, introduced by Oommen and Lanctôt [101]. This family also includes the GPA and its discretized version DPGA, introduced by Agache and Oommen [3]. The GPA algorithm, as shown above, appears to be the fastest continuous pursuit algorithm, and the DGPA is the fastest converging discretized pursuit estimator algorithm. The question of how these LA relate to *associative reinforcement learning* schemes is explained in [141].

2.3.13.2 Parameterized Learning Automata (PLA)

Decentralized learning algorithms (such as the L_{RI}) for a team or a network of automata converge only to local maximum $E[\beta|\mathrm{P}]$. One way to achieve global maximum is to use an estimator algorithm. However, that can have a large memory overhead. Another way to attain to the global maximum is to use an algorithm to impose a random perturbation while updating the action probability vector $\mathrm{P}(t)$ so as to enable the learning process to move from the local maxima. However, introducing a random term directly in the updating equation is difficult for two reasons. First, it is difficult to ensure that the resulting vector after the updating remains a probability vector. Second, the resulting diffusion would be on a manifold rather than on the entire space. These two difficulties can be overcome by parameterizing the action probabilities in terms of some real number, as shown in [141].

2.3.13.3 Generalized Learning Automata (GLA)

These automata provide a way to handle the associative reinforcement learning problem in which the structure of the LA is modified to allow for a context vector input. A single GLA is described by the tuple $\langle X, Y, R, \underline{u}, g, T\rangle$, where X is the set of all

context vectors that can be input to the GLA, Y is the finite set of output or actions of the GLA, R is the set of values that the reinforcement signal can take, g is the probability generation function; and $\underline{u}$ is a vector of real numbers that represent the internal state. T is the learning algorithm that updates $\underline{u}$. For the same state vector $\underline{u}$, the question of choosing different actions depends on the context vector. This enables the GLA to directly handle associative reinforcement learning problems.

2.3.13.4 Continuous Action-set Learning Automata (CALA)

These are automata in which the action set is a continuous set consisting of the real line, instead of being a finite action set. The action probability distribution at t is $N(\mu(t), \sigma(t))$, the normal distribution with mean $\mu(t)$ and standard deviation $\sigma(t)$. The CALA updates its action probability at every instant by updating its normal distribution and mean. Let $\alpha(t) \in \Re$ be the action chosen at t, and let $\beta(t)$ be the reinforcement at t. Instead of a reward probability for various actions, a reward function is used: f: $\Re \rightarrow \Re$ defined by $f(x) = E[\beta(t)|\ \alpha(t) = x]$. The reinforcement to action x is β_x and thus $f(x) = E[\beta_x]$. The objective of the CALA is to learn the value of x at which f attains a maximum.

2.3.14 Multi-Teacher Environment

The field of LA and stochastic learning traditionally deals with the model in which the Teacher (Environment) attempts to teach a Student (LA). In a multi-Teacher environment, an automaton evokes a vector of responses due to multiple performance criteria. In this case, the automaton needs to find an optimal action that "satisfies" *all* Teachers (Environments). The automaton is connected to N Teachers/Environments. A single Teacher/Environment is described by the action set $\underline{\alpha}$, response set $\underline{\beta^i} = \{0, 1\}$ (for a P-model), and penalty probability set $\{c_1^i, c_2^i, \ldots, c_r^i\}$ where $c_j^i = \Pr[\beta^i(t) = 1\ |\alpha(t)=\alpha_j]$. The action set of the the automaton is the same for all Teachers/Environments.

In a multi-Teacher environment, the task of "interpreting" the output vector of the different Teachers is important. In its simplest form, all Teachers can agree on the ordering of the actions where one action would be optimal for all Environments. However, this is not always the case in different applications [147]. One option is that the output of the different Teachers can be summed after normalization. Other methods involve introducing weight factors associated with specific Teachers [9, 84], use AND-Gates [133], or OR-Gates, and additional *If-Then* conditions which forces the system to satisfy all the Teachers simultaneously [147, 148].

Koditscheck and Narendra [58], originally, studied the learning behaviors of fixed-structure automata acting in a multi-random Environment. Thathachar and Bhakthavathsalam [133] then considered the learning behaviors of VSSA in two distinct random Environments. Baba [9] then generalized the model when the Student is learning from a multi-Teacher set. He extended the absolutely expedient learning algorithm [62] and proved that it was absolutely expedient and ϵ-optimal in the general n-random Environment (stationary and P-model), and ϵ-optimal under the non-stationary multi-random Environment.

Ansari and Papavassilopoulos [6] presented a learning algorithm for multi-input multi-output models in which different actions selected by the automaton are given as inputs to multi-Teacher environments. They proved that their algorithm is absolutely expedient and ϵ-optimal in the sense of the average penalty. Baba and Mogami [11] presented a learning algorithm for the hierarchical structure LA operating in the non-stationary multi-Teacher environment, and showed that it converged w.p. 1 to the optimal path under certain type of non-stationary multi-Teacher properties. Their algorithm proved to have a fast convergence. They modified the fast DGPA to be used as a learning algorithm in a hierarchical structure LA, and then compared their algorithm with this modified version of the DGPA. In the non-stationary multi-Teacher environment, their algorithm was 42% faster than the modified version of the DGPA [12].

Ikebo *et al.* [54] used LA to solve the graph partitioning problems, where they used

the S-model LA with a multi-Teacher environment. They, experimentally, demonstrated that their algorithm has some advantages over the most famous partitioning methods, like Simulated Annealing and the Mean Field Algorithm.

2.3.15 Multiautomata Systems

In the the previous Sections, we described the different classifications of single LA models. Systems that consist of many LA can be useful also in solving different and more complex learning problems.

If the different automata are interacting through the Environment, then the multi-automata system is not different from a single automaton scenario [147]. In this case, the Environment reacts to the actions of the multiple automata, and the Environment's response is the result of the combined effect of the action chosen by all automata. On the other hand, if there are multiple automata interconnections and they are interacting directly between each other, then some automata directly affect the actions of others. This behavior is obvious in the hierarchal automata models.

Other models of multi-automaton models are also the feedforward networks, and parallel operations of LA [85, 142]. These different types are described below.

2.3.15.1 Games of LA

This mimics a scenario where N players are involved in a game, in which each player has a set of actions to choose from. For each play of the game, each of the players chooses a specific action from their respective actions set. After that, each player gets a payoff from the Environment, where these payoffs are random. Because the game possesses incomplete information, the players have no knowledge of the probability distribution that determines their payoff for different actions. The problem would be to find the optimal actions of the players. Observe that this can translate to unequal or common payoffs to different players [142].

Each player is represented by an automaton, and the players actions constitute the action set of the automaton. Some of the players in the game could be FALA, while others could be CALA. The reinforcement received by each automaton depends on the actions chosen by *all* automata. At every step in the game, each automata sends an action to the Environment, which evaluates the actions and sends out a feedback, which is the *payoff*, as in the general game theory. The probability vector of actions defines the mixed strategy of a player where each element of the probability vector corresponds to a specific action, or pure strategy.

In the case of a single automaton, there is only one reinforcement, and hence only one function to be maximized. In the case of games of LA, there is a multi-payoff scenario, and hence there are many functions that needs to be simultaneously maximized. There is no obvious way to define what the goal is. Reaching the set of optimal points is one way of defining the objective. These optimal points correspond to action tuples such that the expected value of reinforcement is *locally* maximized.

The formal model for LA games is given in [84], and omitted here as it is not directly related to the main tenets of this Thesis.

Early research in two-person games was initiated in [28, 152]. In a two-person zero-sum game, where one player must lose before the other player can win, there may be more than one equilibrium point with the same payoff. Narendra and Thathachar [84] proved that in two-person zero-sum games, if each automaton is using the L_{RI} scheme, the game converges to the Nash equilibrium[2]. In some non-zero games, and in common payoff games, it is shown that the overall performance improves monotonically if each automaton uses the L_{RI} scheme.

One special case is the game with common payoffs, where all automata receive the same payoff and all of them are trying to maximize the expected value of the payoff. Effectively, the automata are cooperating with each other and behaving as a

[2]The Nash equilibrium is a set of mixed strategies in non-cooperative N-person games with a finite strategy set, where no player has anything to gain by changing only his or her own strategy. Each player's strategy is an "optimal" response based on the anticipated rational strategy of the other player(s) in the game.

team. Although in this case the model is completely decentralized, the automata are cooperating between each other [142].

If the identical payoff game is such that a unique equilibrium point exists corresponding to the global maximum, convergence is almost guaranteed. However, Sastry and Thathachar [142] explained that L_{RI} scheme can learn only a mode of the reward matrix which is a local maximum, i.e., it can always converge to a maximal point. They used the Pursuit Algorithm, which results in both faster convergence and convergence to global maximum. The payoff in this case is a more memory overhead. Also, the Pursuit Algorithm is not a decentralized technique. Other researches have also used the estimator algorithm for the games of LA [78, 139, 140].

Sastry *et al.* [117], using a more general absolutely expedient algorithm, studied the N-person game with multi-payoff. They proved that all stable stationary points of the algorithm are Nash equilibria for the algorithm. These results have been generalized to include a team of both FALA and CALA [109], and to consider nonzero-sum games [84].

It is important to mention that in the area of games of automata, the players are not aware of the mixed strategy used by the other player(s) nor of their previous actions. They do not even have the knowledge on the distribution of the random payoff structure as a function of the combinations of the strategy. The interaction medium between the automata is the payoff function [147].

2.3.15.2 Interconnected Automata

One of the bottlenecks of using a single automaton was that it "can hardly cope with the problems with high dimensionality" [11]. In the case of games of automata, the interaction between the automata is provided by the Environment, where it reacts with a random response which is the combined effect of the decisions of all players. However, the automata are not directly interacting with each other.

Interconnecting automata can increase the potential of learning automata and enable them to be used in more complex systems. However, the very generality of the action set $\{\alpha^j\}$ and the Environment response $\{\beta^j\}$ "makes the definition of meaningful interconnection rather difficult" [84]. There are mainly two models of interconnection automata: synchronous and sequential.

In a synchronous model, all automata react to their respective Environments synchronously. In this model, the response from the Environment associated with automaton A_i is denoted by $\widetilde{\beta}^i$ and the input (to A_i) by β^i, where the latter depends on the response from all automata. The synchronous model can be viewed as a game of automata with particular payoff structures. Figure 2.2 shows a basic synchronous model where two automata Environment pairs are connected in a feedback configuration.

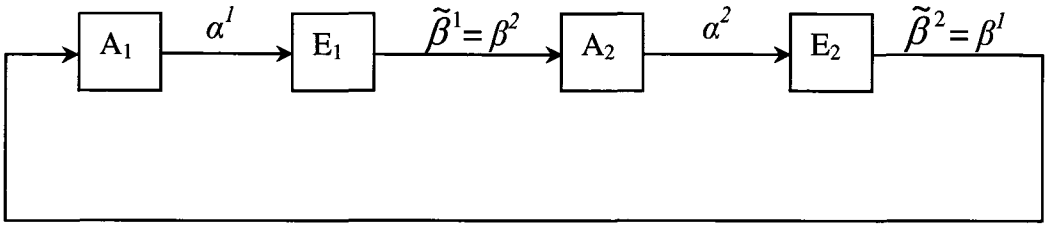

Figure 2.2: Basic Synchronous Model [84].

In the sequential model, only one automaton is acting at each time step, and depending on the action of that automata, the next automaton is selected for the next time step. This sequential model of multi-automata can be viewed as a network of controllers where control is passed from one node to the next [85]. The goal of the whole group is to optimize the overall performance criteria [147].

One of the main structures of sequential models is the hierarchal form, which has been extensively studied by many active researchers. Hierarchical Structure Learning Automata (HSLA) was introduced by Thathachar and Ramakrishnan [135]. Figure 2.3 shows one example of a hierarchical model where each automaton has two actions. In this example, Thathachar and Ramakrishnan developed a group of several automata in multiple levels, each with few actions instead of one automaton with

many actions. The algorithm was proven to be ϵ-optimal but it requires each level to know the action probabilities used at the next higher level. To get faster convergence, a heuristic approach was also used to select hierarchical structures with minimal computational effort while maintaining equivalency [76].

Narendra and Parthasarathy [81] proved that HSLA can still be absolutely expedient when different rewards are used at the various levels of the HSLA. Papadimitriou [104] used a hierarchical discretized pursuit non-linear learning automaton to position the actions on the leaves of the hierarchical tree and proved that it is ϵ-optimal for all non-stationary Environments.

As mentioned in Section 2.3.14, Baba and Mogami [10, 11] introduced a new learning algorithm for hierarchical structure learning automata in non-stationary multi-Teacher Environment. They proved that their algorithm converges w.p 1 to the optimal path when operating in a general non-stationary multi-Teacher Environment.

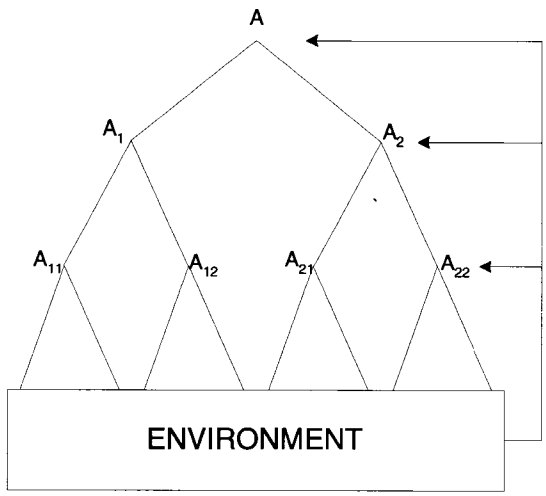

Figure 2.3: Hierarchy of Automata [85].

2.3.15.3 Feedforward Networks

In this model, the learning system, in the form of a network, attempts to learn the best action associated with each context vector [142], where each node of the network is a

team of LA. Each node can also hold a more complex structure such as hierarchical organization of LA teams. Figure 2.4 shows a feedforward network where each unit is a team of LA. The feedforward model considers the computational advantage of a team of FALA over a single FALA. The feedforward networks enables the handling of more complex learning problems.

However, with this advantage of using the L_{RI}, the team converges to a local maximum instead of the single LA that converges to global maximum. Global convergence algorithms could be used to converge to global maximum, however, it would cost longer convergence time and higher computational efforts [142].

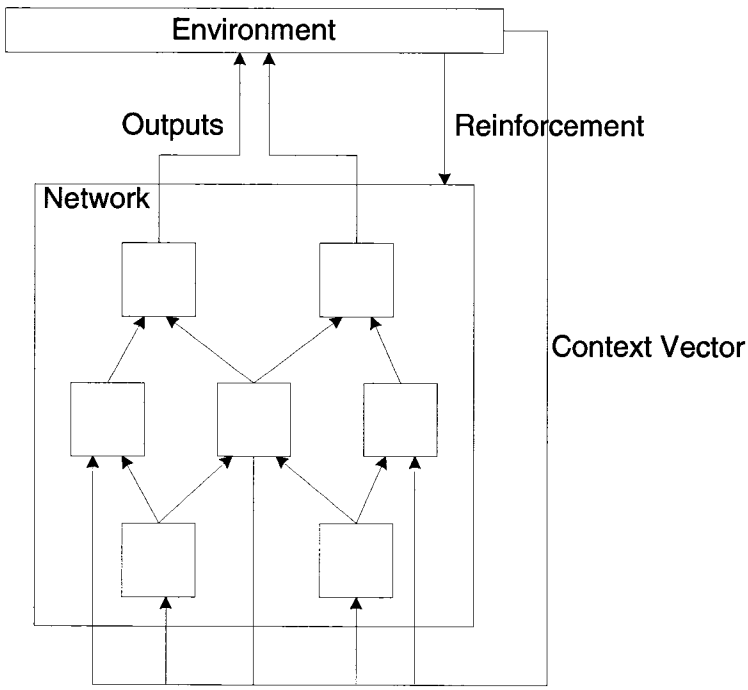

Figure 2.4: A Network of Learning Automata. Each of the boxes corresponds to a unit which is a team of FALA [142].

2.3.15.4 Parallel Operation of Learning Automata

One question always encountered is how to speed the convergence of LA without compromising accuracy [132]. One way of speeding the convergence is by using a number of parallel LA instead of a single LA. This provides faster convergence by updating action probabilities based on several responses. This would provide less expected error than those based on a single response [142].

Parallel LA can be used in some applications, for example in pattern recognition, where multiple parameter vectors are received simultaneously. Other applications can not use parallel operation, for example network routing, where the action corresponds to a route and the simultaneous choice of several actions is not feasible.

Thathachar and Arvind [132] showed improvements in the convergence speed by utilizing parallelization. They implemented a parallel algorithm, namely, a parallel version of the L_{RI} algorithm, and proved that their algorithm was ϵ-optimal for a small learning parameter, λ. As mentioned in Section 2.3.14, Ansari and Papavassilopoulos [6] also presented a parallel algorithm for multi-input multi-output models in which different actions selected by the automaton are given as inputs to multi-Teacher environments with identical characteristics. They proved that their algorithm is absolutely expedient and ϵ-optimal in the sense of the average penalty.

2.3.16 Applications of LA

Learning automata have been used in systems that have incomplete knowledge about the Environment in which they operate [3]. Narendra and Thathachar [84] identified four "desirable" features of a system in which LA can be used, as follows:

- The system is complex and contains a high degree of uncertainties, where the uncertainty cannot be modeled mathematically.
- The system must be "amenable to distribution control". At every stage, the

decision maker chooses an action, from a finite set of actions and decisions.

- Every decision maker receives a feedback by some random realization of a performance criterion.
- The system must behave in a way in which a small improvement in performance will result in a significant economic return.

These criteria define, to a large extent, how attractive LA-based schemes can be when applied to practical applications. In his pioneer work, Tsetlin [145] attempted to use LA to model biological learning. There are varieties of applications that use LA in different fields. They have been used in telecommunications and telephone routing [72, 83], image data compression [47], pattern recognition [97, 15], graph partitioning [96], object partitioning [94, 102], vehicle suspension [69], path planning for manipulators [86], and vehicle path control [148].

We propose to use them to design Intelligent Tutorial-*like* systems.

2.4 Tutorial Systems

Conventional human tutoring is a tried and tested educational method that can result in substantial improvements in student performance. Producing a system that can emulate that interaction is a worthy (but not easily attainable) goal. Two of the most important aspects of an effective tutor are:

- It possesses the knowledge necessary to solve the problems presented to the student.
- It contains the knowledge about how students make errors.

An effective tutor can identify student errors, and explain how to arrive at a correct solution. The tutor should automatically adapt to the student's needs. Atkinson *et*

al. [7] defined learning as "a relatively permanent change in behavior that occurs as the result of practice".

The teaching/learning process involving a group of students is different from the corresponding teaching/learning process applicable for a single student [92]. If a tutor is teaching a group of students, s/he explains the subject matter and evaluates the learning by the behavior of *some* of these students. In the literature, this evaluation is performed by, for example, making queries, accepting and analyzing answers, providing examples and counter-examples, writing and drawing on the "blackboard", or showing slides, and/or by video inputs. Students understand the subject matter in different degrees. Additionally, the students themselves can "learn" from each other. On the other hand, if the educational process is centered on a single student, this process has to consider only his/her needs, and the teacher can control the entire learning process in a straightforward manner.

2.4.1 Computer Aided Instruction (CAI)

Using computers in tutoring systems started in the field commonly known as Computer Aided Instruction (CAI). In the 1960's, researchers developed a number of CAI tools that were generative. These programs generated problems designed to enhance student performance in arithmetic and vocabulary-recall [146].

By the late 1960's and 1970's, a number of systems were developed to provide drill and practice that, in turn, selected problems appropriate to the student's overall performance [129]. These systems have been called "adaptive" and were the first to try to model students. In these systems, models of the student "were based more on parametric summaries of behavior than explicit representations of his knowledge" [126]. Because of the simplicity of the task domain used, these techniques proved to be effective for instructional uses.

2.4.2 Intelligent Tutoring Systems (ITSs)

Intelligent Tutoring 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 [92]. 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. ITSs involve the acquisition and encoding of at least these three types of information [29]:

- Instructional material.
- Domain knowledge (conceptual, inferential, and procedural).
- Pedagogical knowledge (e.g., lesson-planning and coaching strategies).

By the late 1970's and early 1980's, research in ITSs focused on supportive learning environments intended to facilitate *learning-by-doing*, which involved transforming factual knowledge into experiential knowledge. By that time, ITSs took the form of computer-based problem solving monitors, coaches, laboratory instructors and consultants. They were also experimented in a variety of domains [126].

By the late 80's, researchers thought of ITSs as an attractive research subject that were to yield good rewards. Wenger [154] proposed a new discipline of ITSs that combined the fields of AI, cognitive science, and education as follows:

> *"... consider again the example of books: they have certainly outperformed people in the precision and permanence of their memory, and the reliability of their patience. For this reason, they have been invaluable to humankind. Now imagine active books that can interact with the reader to communicate knowledge at the appropriate level, selectively highlighting the interconnectedness and ramifications of items, recalling relevant information, probing understanding, explaining difficult areas in more depth, skipping over seemingly known material ... intelligent knowledge communication systems are indeed an attractive dream".*

Brusilovsky and Peylo [22] summarized the major technologies in intelligent tutoring systems into three categories:

- **Curriculum sequencing technology**: The goal of the systems is to provide the student with the most suitable individually planned sequence of topics to learn, and the learning tasks (examples, questions, problems, etc.) to work with. It helped the student find an "optimal path" through the learning material.
- **Intelligent solution analysis**: These systems deal with students' solutions of educational problems (which can range from a simple question to a complex programming problem). They can analyze the errors made by the students and discover missing or incorrect pieces of knowledge due to these errors. They also provide extensive feedback, and provide updates to the student model.
- **Interactive problem solving support**: The goal is to provide the student with intelligent help on each step of problem solving - from giving a hint, to executing the next step for the student.

An ITS mainly consists of a number of modules, typically three [39, 88], and sometimes four, when a communication module (interface) is added [155]. The former three modules are the domain model (knowledge domain), the student model, and the pedagogical model, (which represent the tutor model itself). Self [120] defined these components as the tripartite architecture for an ITS – the *what* (domain model), the *who* (student model), and the *how* (tutoring model). Some systems add more components to these basic fundamental components. For example, a supervisor unit that supervises the function of the ITS and interacts with the other components of the ITS to controls the functionality of the whole system [108] has also been recommended. Figure 2.5 depicts a common ITS architecture. Each of these models are explained below.

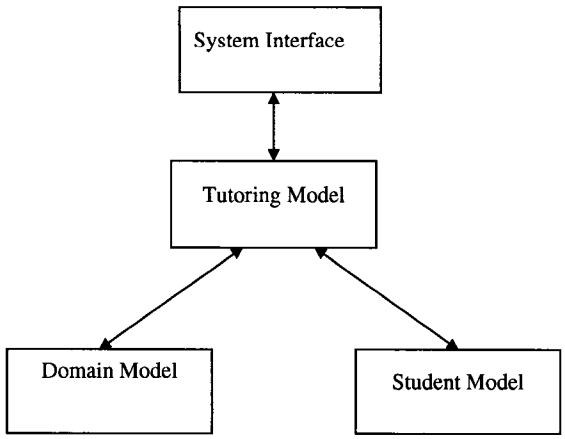

Figure 2.5: A Common ITS Architecture [23].

2.4.2.1 Domain Model

The domain module is a control center that encompasses the entire domain knowledge, which generates instructional content and evaluates student's performance [45]. The domain knowledge contains principles, procedures and techniques for conceptual modeling and statistical analysis. It consists of rules that determine the performance of the learner and provides appropriate responses. There are three types of rules [8]:

- **Student rules:** Such a rule uses the current task to generate the next operation. It relates the current tasks to the classified structured tasks in its knowledge base.
- **Operational rules:** These rules specify operational task behavior that must be satisfied before the intelligent help is triggered.
- **Procedural rules:** These are used to specify the tutorial operation to be executed when the learner has satisfied certain conditions.

The domain knowledge usually consists of declarative and procedural knowledge. It can also contain heuristic knowledge [111]. Each of these is explained below as:

- **Declarative knowledge:** This defines terms with their attributes, and defines the relationship of the terms to each other.
- **Procedural knowledge:** This consists usually of rules (i.e., how to do things), which aid the problem-solving process.
- **Heuristic knowledge:** This can be seen as the experience that an expert would use for solving problems. Heuristics can be used for faster and sorted analyses.

Most ITS models consider how an expert would use the knowledge as a part of the domain knowledge. However, there are few models that split this knowledge into separate portions using an expert model [18]. They consider that an expert model is a model of how an expert would use the knowledge contained in the domain knowledge. An assessment of a student's solution could be compared with that of an expert given the same domain knowledge.

Expert knowledge can be encoded in three different categories: completely opaque or "black box" model, fully transparent or a "glass box" model, and the cognitive model [5]. In the black box model, the system encodes the knowledge without codifying the underlying human intelligence. The model assesses the student's performance by generating the correct input-output behavior over a range of tasks specified in the domain. SOPHIE-I [21], which troubleshoots electronics, is considered an example of the black box model.

In the glass box model, the knowledge domain is modeled as an expert system which tries to reproduce the problem-solving behavior of human experts [44]. An example of this is GUIDON [32], based on MYCIN, an expert system for diagnosing bacterial infections. This model requires more implementation efforts than the black box model, but it is also more effective.

The cognitive model simulates expert knowledge, as well as the way students use this knowledge. It contains a psychological component where the system tries to

teach the student in a human-like tutor. Although this type is the most demanding approach of building expert knowledge, it is also the most rewarding [5].

2.4.2.2 Student Model

Human tutors can easily judge the student's answers in the context of his/her assumed level of understanding and past learning behavior, thus effectively adapting their instruction to the student's competence and abilities [112]. No intelligent communication can take place without a certain understanding of the recipient. Thus, along with the idea of explicitly representing the knowledge to be conveyed, the idea of doing the same with the student came in the form of a student model [154].

The student model may be defined as a sequence of actions performed during the tutorial, and consists of sets of possible behaviors [8]. It is used to determine the student's current knowledge status, his conceptions, and reasoning strategies [45]. The student model representation and reasoning formalism should be able to deal with uncertain and vague knowledge. Also, heuristic knowledge is required to make evaluations [48].

Kass [56] defined three areas in student modeling that are relevant to this research, and which need consideration:

- What information about a student needs to be modeled?
- How is this information represented?
- How can a student model be built?

There are different types of information that need to be stored in a student model. Prentzas *et al.* [108] identified four types of items which a student model may contain:

- Personal data: Used for identifying the student.

- Interaction parameter: Records the interaction between the student and the tutorial system.
- Knowledge and concepts: Defines what the student knows about the domain knowledge.
- Student characteristics: These are mainly multimedia type preferences, knowledge level of the sub-domain and the whole domain, concentration level, ITS-using experience, and the desired level of presentation.

Sison and Shimura [124] defined a student model as a qualitative representation of the student's knowledge about a particular domain. However, they considered the student model as one of three elements that compose *student modeling*. The two other elements were the student behavior and the background knowledge. The student's behavior refers to his/her observable response to a particular stimulus in a given domain. The background knowledge includes the correct facts, procedures, concepts, principles, and strategies of the domain. It may also include the misconceptions and errors that can be made by a population of students in the same domain.

There are mainly three different classes of students who use tutorial systems [121]: the *non-cooperative*, those who act in a passive way and even try to frustrate the system; the *cooperative*, those who flow where the system leads them, but do not necessarily know where to go; and the *pro-active*, those who know their objectives and search proactively to achieve their goals. The teaching methodology for each type is different and the teaching environment should take this difference into consideration when designing the student model.

Knowledge Representation in the Student Model:

Sleeman and Brown [126] classified the knowledge representation for student models as being one of three types: overlay, differential, or perturbation.

- **Overlay:** This model, originally developed by Goldstein [43], assumes the modeling of the student knowledge as a subset of an expert's knowledge (Figure 2.6).

The overlay model cannot recognize if the student's knowledge differs from that of the expert because the student does not have that knowledge, or because he/she follows different strategies from the expert. Some models that use the general overlay model are SCHOLAR, Geography tutor [35], and GUIDON [32]. The main problem with this model is that the assumption of the student's knowledge merely being a subset of the expert knowledge may not be the case. Novice learners usually approach the task of problem solving in a manner different from that of the experts [52].

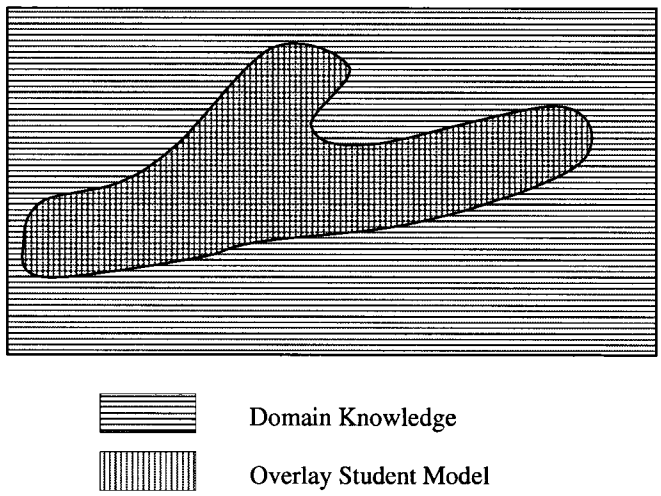

Figure 2.6: An Overlay Student Model [56].

- **Differential:** This model abstracts how the student's behavior is critically different from that of an expert (see Figure 2.7). It was introduced by Burton and Brown [25] and required two tasks. The first task is to evaluate the current student action relative to the possible alternatives actions that an expert might have made in the exact same conditions. The second task is to determine the skills gained by the student's action as well as each of the "better" moves of the expert. Examples of differential models are WEST, an arithmetic tutor [26], and GUIDON2, a medical diagnostic system [33]. Although the differential model

is not so strict about the knowledge of the learner, it still has the same problem as the original overlay model. Since it assumes that the student knowledge is essentially a subset of the expert, the student model remains incomplete [52].

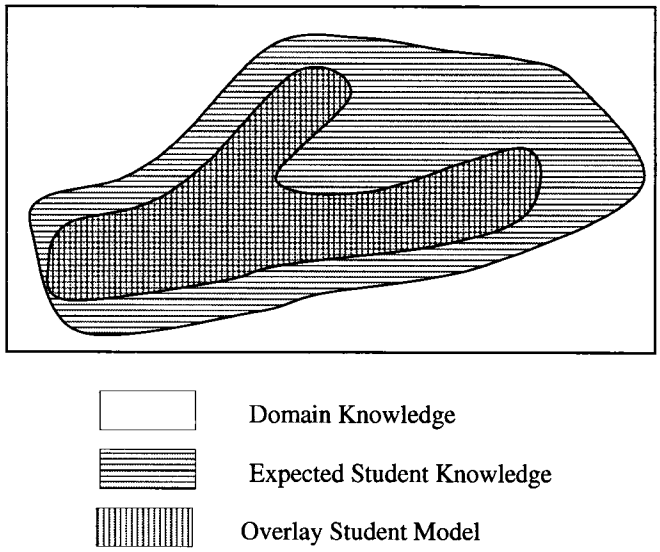

Figure 2.7: A Differential Student Model [56].

- **Perturbation:** This model represents student misconceptions as variants of the procedural structure of the experts' correct skill. A perturbation model combines the standards overlay model with a representation of faulty knowledge. The student's knowledge is not considered a mere subset of the expert's knowledge; it is different in quantity and quality [52]. The perturbation model is also called a *buggy model* [26]. A common technique for implementing the perturbation model is to represent the expert knowledge and then add misconception knowledge [52] (see Figure 2.8) to that knowledge. Examples of the perturbation model are DEBUGGY, a system used to evaluate a learner's subtraction performance [24]. In models of errors, bugs analysis, and diagnostics, libraries with possible bugs are added on the top of the domain knowledge. Because adding these bugs is a very work-intensive task, Ohlsson and Langley

[91] use machine learning to register bugs in the system. However, "neither the bug library technique nor the machine learning approach is currently used extensively in instructional computing systems" [90].

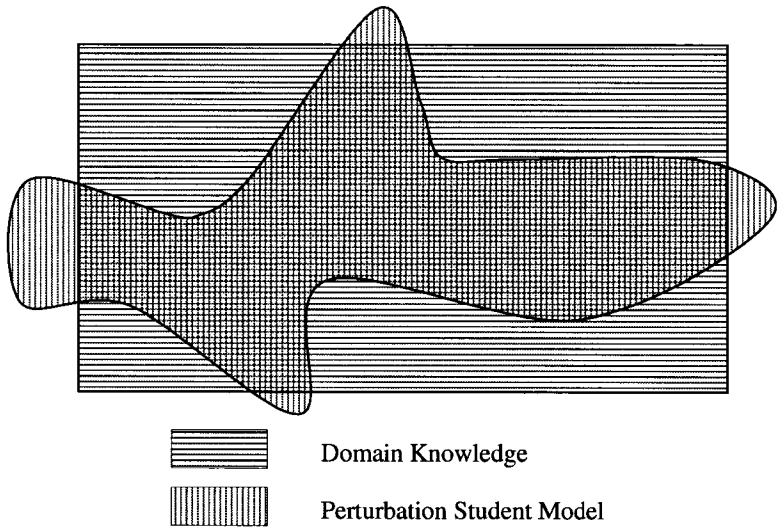

Figure 2.8: A Perturbation Student Model [56].

Theories of bugs emerged for developing and representing bug libraries. Wenger [154] defined three types of theories of bugs: enumerative, reconstructive, and generative. Table 2.4 presents a classification of some existing examples of theories of bugs. Since actual systems often combine characteristics from more than one type, the classification is done in two passes: the main classification (rows) is refined with additional distinction (columns) along with some dimensions. Hence, pure examples of each type are on the diagonal.

Enumeration theories of bugs enumerate bugs in the form of catalogs or libraries of commonly observable deviations based on empirical analysis of students' errors [52]. Although these bugs could be purely descriptive, they are usually represented as pieces of a process model such as sub-procedures, mal-rules, or incorrect plans.

	Enumerative	Reconstructive	Generative
Enumerative	ACTP: extensionally defined list of observable errors	DEBUGGY, LMS: combination of enumerated bugs that reconstructs observed errors	MENO-II: enumerated errors recognized and attributed to enumerated misconceptions
Reconstructive	PROUST: reconstructs design intentions using a library of buggy plans	ACM, PIXIE, ADVISOR: reconstructs bugs from a language of neural primitives	Young & O'Shea: incorrect procedures reconstructed with manipulations that explain the nature of bugs
Generative	Bonar & Soloway: library of abstract bug generators to explain the origins of observed errors	REPAIR: generates bugs by replying the inventive handling of impasses	REPAIR/STEP, Matz: reduction of the occurrence of bugs to mislearning

Table 2.4: Types of Theories of Bugs [154].

Reconstructive theories of bugs escape the limitations of fixed libraries by reconstructing internalized bugs on the basis of observed errors, usually in a data-driven, bottom-up fashion. In a pure form, it is different from enumeration theories in that they use building blocks that are neutral in terms of correctness.

Generative theories of bugs are similar to the reconstructive theories in that they possess a language capable of expressing surface bugs, and define a space of variants of the target knowledge. However, they can explain bugs in terms of their generation from an underlying cognitive model [52].

In addition to the three main types of knowledge representation in the student models defined above (overlay, differential, and perturbation), there are also other types that are used in student models:

- **Constraints-Based:** In this case the domain knowledge is represented as a set of state constraints. Constraints define sets of equivalent problem states [77]. Students are observed during the process of solving problems in an ITS, and their student models collect the constraints that are and aren't violated. The

latter correspond to the students' incorrect and correct solutions, respectively [156]. Example of a constraint-based student model is the SQL-Tutor [130], which the authors argue is "extremely effective".

- **Bayesian Network:** Bayesian belief networks (BBNs) are used to deal with situations that require reasoning under uncertain information [106]. BBNs have been applied in several areas of user modeling, such as plan recognition, knowledge assessment, and prediction of user responses [55]. A BBN allows for reasoning about a situation by assigning probabilities of truth to beliefs and inferences regarding that situation [156]. Example of an ITS that incorporates a Bayesian network in the student model is ANDES [34], which help students learn Newtonian physics.

- **Fuzzy Logic:** This model is considered to be more practical and less precise than the one that incorporates Bayesian networks [55]. Fuzzy logic is used to describe the teacher's subjective linguistic description of a student's behavior (e.g. the student draws a vector after a *long* time; the student answered *enough* questions in the pre-test) [127]. A fuzzy logic system first defines an arbitrary set of possible degrees of membership in a class such as: [False, Unlikely, Likely, True]. This scale of set membership is then applied to the possible members of the sets that are defined in a system [156].

- **Hybrid approaches:** Theses approaches combine two or more approaches. They are developed in an effort to create improved representations [108]. Some of these approaches are neuro-symbolic or neuro-fuzzy representations. For example, WITNeSS [87] uses neuro-fuzzy representation in student modeling.

McCalla [74] classified student modeling from a cognitive perspective to acquire information into two types of modeling: external and internal student models. In external modeling, the system is designed to infer the cognitive processes of typical students. The system carries out cognitive modeling of the student. On the other hand, in internal student modeling, the system maintains an explicit model of the cognitive

process carried out by the student. In this case, the system takes into account the actual cognitive process of the learner.

The internal model is more desirable than the external model, because it makes the system more adaptable to each student. It allows the system to store relevant knowledge about the student and to use this knowledge to adapt to individual student needs. In the external model, decisions are made only on the bases of a snapshot of observed student behavioral patterns [52].

Student Models Tasks:

According to Wenger [154], student models have three tasks. Their first task is that they must collect information about the student. This can take the form of inferring unobservable aspects of the student's behavior in order to produce an interpretation of the student's actions. It can also reconstruct the knowledge that has given rise to these actions.

Their second task is that they must span both correct and incorrect knowledge to represent the system's expertise. This could be solved by collecting information about likely errors and misconceptions for a given domain. The system then selects, from its models of correct and incorrect knowledge, elements which are the ones that account for the student's behavior.

Finally, they must account for the data by analyzing the specific data made available to the system. This diagnosis would help in terms of selecting optimal pedagogical strategies for presenting subsequent domain information to the student. One important consideration is how the diagnostic process deals with the presence of *noise* in the data. Such noise characteristics exist, inasmuch as the model is almost a simplification of the complex process of human reasoning, where students are almost never perfectly consistent.

Functions of the Student Model:

From an analysis of student models, Self [119] determines six groups of functions

that a student model can have:

- Corrective function: The system must be able to find incorrect knowledge of the student and correct it.
- Elaborating function: If the system determines that the learner's knowledge is correct but incomplete, it suggests actions to the student by comparing the expert model with the student's current state of knowledge.
- Strategic function: The analysis of the learner's performance might lead to other learning strategies or higher level knowledge.
- Diagnostic function: The system must be able to find out the learner's ideas, analyze the student model, and decide on certain actions by itself.
- Predictive function: The system must be capable of simulating the student's behavior so as to be able to predict the direction of further actions.
- Evaluative function: The system must be able to reconstruct the "learner's learning process", and accordingly evaluate the learner's performance.

Verdejo [151] considered an extra seventh group to the above. He considered the "communicative" function to support a mixed-initiative style of interaction.

Barriers for Student Modeling:

Holt *et al.* [52] defined some barriers to student modeling that result from the problem of inferring knowledge from the student's behavior. These barriers are:

- The environment contains a large amount of uncertainty and noise.
- The student's inference may be unsound, and may be based on inconsistent knowledge.
- Constructing explanations from student behavior are computationally intractable.

- Students are creative and inventive; interpreting their behavior requires extensive sophistication.

2.4.2.3 Pedagogy (Tutorial) Model

The pedagogy model specifies how the student should be taught [8]. It contains a series of specifications about what instructional material should be presented, including the presentation method and timing [45]. Its functionalities can be summarized as "what to present, when, and how", and sometimes also as "when to interrupt" [70].

There are usually two approaches for presenting the learning material in this module: the Socratic method and the Coaching method [57, 154]. In the Socratic method, the tutor uses questions to teach and evaluate the performance of students, and to guide them through the process of discovering their own misconceptions. An example of a Socratic method tutor is the WHY tutor [128], which teaches students about the causes of rainfall. The Coaching method provides the student with an environment in which to engage in activities so as to learn related skills and general problem-solving skills.

Recently, rather than using these two methods, some research has focused on complex environments that promote interaction and feedback. These environments aim at improving the student's understanding by formulating ideas, discovering relationships, drawing conclusions, and realizing misconceptions [45].

After deciding the higher-level strategy, low-level issues on the contents of the material presented to the student must be considered. Beck *et al.* [18] defined some of these issues as follows:

- Topic selection: This involves selecting new material or old material for reviewing. The tutor must examine the student model in order to decide what will be the next topic that has to be presented to the student.
- Problem generation: After the selection of a topic, a problem must be generated

or selected to be solved by the student. The problem should be appropriate to the student's ability, which can be determined from the student model.

- Feedback: The type and style of feedback is particularly important when the student makes an error. The advice given to the student should be dependent on his/her ability and level. Too little feedback can be frustrating, while too much feedback can interfere with the learning process.

Tutorial systems need to decide how to correct students' misconceptions. The question involved is whether the tutorial system tells the student his/her errors explicitly, or whether it lets him/her discover the errors on their own. Burton and Brown [25] recommended that the tutorial system should intervene only when the student is likely to fail to discover his/her errors, or misconceptions, and argues that "every time the coach tells the student something, it is robbing him of the opportunity to discover it for himself".

Mathan and Koedinger [73] mentioned that immediate feedback has been criticized for mainly two reasons. The first is that immediate feedback from an ITS is qualitatively different from that offered by a human tutor. For example, a human tutor can detect if an error can provide a learning opportunity, using which he can guide the student through the error detection activity.

The second reason is that some empirical studies have highlighted the benefits of delayed feedback. Immediate feedback, although not always desired, may be chosen for domain or technical reasons [112]. An example is the system Geometry in [110], and the LISP tutors, which immediately tells the students of their errors. Here, the assumption is that errors in this domain are ambiguous, and erroneous solution paths would be a waste of effort. However, empirical results suggest that skill acquisitions are "most efficient with immediate feedback" [73].

2.4.2.4 User Interface (Communication Model)

User interface, or the communication model, allows communication between the student and the other aspects of the ITS. The communication model is the model that appears to be "intelligent" to the student [111]. The user interface processes the flow of communication (in and out) between the system's internal representation, and an interface language understandable to the student [154]. User interfaces for ITSs should be robust, flexible, and easy to use and understand [20].

The user interface is an important component in the success of an ITS. Without a proper interface, students would not be able to get the full benefit of the system. It would be a mistake to consider the role of the interface as secondary when compared to the other components of the tutoring system. The interface can change the presentation to the student to be either more or less understandable, which in turn, is crucial for the student to accept the system [154].

Two classical approaches for user interface designs are to use a natural language interface or a graphical interface with buttons, icons, etc. Using a natural language as an interface does have the advantages of being well understood, but the language may not be self-explanatory. Natural language processing still has some restrictions which decrease their benefits. The requirements for the natural language interface to an ITS are efficiency and friendliness [21]. Graphical interfaces could improve the capabilities of the user interface by including features like pull-down menus, graphical representations and attractive screen designs. However, it may draw the student's attention from the subject to the interface itself, e.g. with too many icons, or having too many features.

In general, there are usually two types of user interfaces in tutorial systems [111]. The first type is the system that follows the instruction-concept. In this type, the student can move freely within the tutorial system. The system has a lot of interaction with the student to guide him/her through units and exercises. The other type is the microworld-concept (or construction-concept), which is more free and does not

interfere much with the student's actions.

The ultimate goal of an ITS is to replace human teachers. However, Wenger [154] pointed out that adaptability in the best tutorial systems was rather coarse when compared to that of a human tutor:

> *"The anthropomorphic view that more intelligence for systems means more human-like capabilities can be as much of a distraction as it is an inspiration. Indeed, the communication environment created by two people and that created by a person and a machine are not likely to be the same. The terms of the cooperation required for successful communication may differ in fundamental ways. Hence, computational models of knowledge communication will require new theories of knowledge communication, as computer-based systems evolve and as research in artificial intelligence and related disciplines provides more powerful models".*

2.4.3 ITSs and Agents

Traditional ITSs are based on a rigid interaction between the tutorial system and the student using the system. Using intelligent agents is one of the ways by which one can lessen this rigid interaction. Using agents in ITSs "enable more natural interaction that are closer among students and the tutor system, where the initiative of the interaction is usually shared between the system and the student" [115].

Using agents in the design of an ITS can lead to more versatile, faster, and less expensive systems [121]. The goal of tutoring agents is to "perform a given tutoring function for the student's benefit" [67]. Some of these functions include presenting or explaining a subject element, giving an example, or answering a student's question. The three main components of an ITS (student model, domain knowledge, and the tutoring model) can be built in the form of intelligent agents [40].

Baylor [16] classified the agent-student relationship into three possibilities. The agent, or multiple agents, can serve as a personal assistant, a pedagogical expert/mentor, or as a learning companion as explained below:

- Personal assistant: The agent facilitates and automates certain learning-related tasks. It could assist the system or the student.
- Pedagogical expert/mentor: The agent can monitor and evaluate the timing and implementation of teaching interventions (e.g., help, feedback). It can also exchange information with the student in order to adapt the presentation to the student according to the student's ideal model [115].
- Learning companion: The agent can function as a learning companion in which the agent learns together with the learner. This type is effective if it can replicate the behavior of a novice with whom the student can work.

Towns *et al.* [143] classified the research on pedagogical agents into two types. The first is the research on pedagogical agents that can facilitate the construction of component-based tutorial system architectures and communications between their modules, provide pedagogical strategies, reason about multiple agents in learning environments, and act as co-learners. The second is the research in which pedagogical agents can behave in a lifelike manner. Santos *et al.* [115] defined the main goal of the pedagogical agents as to help the student learn effectively, and to enhance the quality of the learning. In order to do so, these agents can:

- Guide the students during their interaction with the system.
- Monitor students' activities, and provide some help.
- Model students' profiles by recording necessary information.
- Motivate the students to learn.
- Add interactivity to the system, providing students a tutor "friend" who will help them.

Frasson *et al.* [41, 40] have shown that ITS agents have to be reactive, adaptive, instructable, and cognitive. *Reactive* agents can provide response to stimuli without

reasoning, as well as provide handling situations that requires planning, prediction, and diagnostics capability. *Adaptive* agents can adapt their perception of situations and modify their decisions by choosing new reasoning methods. *Instructable* agents can dynamically receive new instructions or algorithms to improve their learning. *Cognitive* agents improve themselves and rely on their capabilities to learn and discover new facts or improve their knowledge.

Lelouche [67] defined three types of agents in tutorial systems. The first are *tutoring agents*, which interact with a student and perform real tutoring functions. Some of these agents would be the problem solver, the domain presenter, the domain assessor, and the exerciser. The second type consists of *service agents,* which may or may not interact with a student. It performs functions on behalf of other agents. This can be the problem selector, the topic chooser, and the question asker. The third type consists of *mixed agents,* which can function either way depending on the circumstances. Examples are the problem step solver and the explanation provider.

Frasson *et al.* [41] designed a tutorial system based on the architecture of multi-agents to support learning using a distribution strategy based on three agents. The three agents are the *tutor* who supervises the learning session, the *troublemaker* who checks and improves the student's self confidence by deciding to give correct solutions or wrong information, and the *artificial learner* who synchronizes the student's activity with the other two agents.

2.4.4 ITSs and Machine Learning

Machine Learning is the ability of computers to reason and make predictions about situations they have not previously encountered. Machine learning can be roughly classified into three types according to the primary approach that they take to inference, induction, deduction, and analogy [123]. Holland *et al.* [51] defined induction as "all inferential processes that take place in the face of uncertainty". Induction is concerned with inferring knowledge from an incomplete set of observations. It is

inherently uncertain because the resulting knowledge is based on incomplete information. Deduction learning (or compilation), on the other hand, works on existing facts and knowledge, and deduces new knowledge from the old knowledge. Assuming that the exiting knowledge is complete and correct, deduction therefore guarantees inference and the truth of this inference. Analogical learning can be viewed as a combination of the first two types.

Machine learning algorithms can also be classified as supervised learning or unsupervised learning [30]. In supervised learning, training examples consist of pairs of input objects and desired output. The task of the learning algorithm is to learn how to predict the output values of new examples, based on their input values. In unsupervised learning, training examples contain only the input objects with no explicit target output. The learning algorithm needs to generalize from the input patterns to discover the output values.

Almost every major approach to symbolic machine learning has been applied to student modeling [124]. This includes supervised induction (e.g., ACM [65] and ASSERT [14]), unsupervised induction (e.g., conceptual clustering in MEDD [122]), or compilation (e.g., a form of explanation-based generalization in MALGEN [53]).

Machine learning is also used in different parts of ITSs. It is used in the construction of background knowledge [131]. Beck *et al.* [19] used machine learning to improve tutoring strategy. Sison and Shimura [124] believed that analogical learning is more appropriate for learning-level analysis, while reinforcement learning, is apparently more appropriate for tutoring. Reinforcement learning can be used to train an agent to comply with the needs of a student [71].

Also, various web-based Tutorial systems have been proposed [158, 153, 4], but as they are outside the scope of this Thesis, we shall not explain them here.

2.5 Conclusions

In this Chapter, we presented a review of the state of the art research in the areas that are used in this Thesis. First, we introduced various families of Learning Automata, their classifications, and a survey of the up-to-date research results. LA will be the learning tool that we use to design our families of Tutorial-*like* systems. Then, we discussed Intelligent Tutoring Systems and their models, as well as the new research in this field. This Chapter established the foundations for our proposed solution to Tutorial-*like* systems, which will be presented in the subsequent Chapters.

Chapter 3

Proposed Model

3.1 Background

Traditional Tutorial systems are cumbersome and often difficult to implement. As seen in the last Chapter, research in Tutorial systems has focused on Intelligent Tutoring Systems (ITS). These systems are usually composed of four models: the interface model, domain model, student model, and tutoring model. They attempt to customize the tutoring experience according to each Student's needs.

As highlighted earlier in this Thesis, we are presenting a new philosophy to design and implement Tutorial-*like* systems using stochastic LA. The proposed system will be composed of an ensemble of LA modules, where each entity can improve its behavior based on the response it receives from its corresponding Environment. Thus, the system also uses LA to model the Student, the learning achieved by the Teacher, the learning achieved by each Student by interacting with the Teacher, and the learning accomplished by the School of Students by interacting between themselves.

This Chapter[1] presents an overall global view of our paradigm. First of all, we

[1]The initial results of this Chapter were presented as a Plenary/Keynote talk at SummerSim'06, the *2006 Summer Simulation Multiconference* in Calgary, July/August 2006 [100].

will introduce Tutorial-*like* systems, and highlight the main differences between them and "traditional" Tutorial systems. Thereafter, we shall present and describe each module in our Tutorial-*like* system. This Chapter is purposefully brief because we will be developing all of these modules individually in the following Chapters.

3.2 Previous Solutions

As seen earlier, using machine learning in improving tutoring systems was the study of a few previous researches. Frasson *et al.* [41] designed the main ITS components (student model, domain knowledge, and the tutoring model) in the form of intelligent agents. Lelouche [67] used a collection of interacting agents to represent the original modeling of the tutoring knowledge in an intelligent educational system. Legaspi and Sison [66] modeled the tutor in ITS using reinforcement learning with the temporal difference method as the central learning procedure. Beck [17] used reinforcement learning to learn to associate superior teaching actions with certain states of the student's knowledge. Baffes and Mooney implemented ASSERT [14], which used reinforcement learning in student modeling to capture novel student errors using only correct domain knowledge. Our method is distinct from all the above.

3.3 Tutorial-*like* Systems

Our entire research will be within the context of Tutorial-*like* systems. First of all, we should mention that Tutorial-*like* systems share some similarities with the well-developed field of Tutorial systems surveyed above. 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 – it is stochastic. Secondly, from the *psychological* perspective, the Teacher need not consider the cognitive state of the Students, although he may consider the Students' learning capabilities. Thirdly, the Teacher does not *initially* possess any knowledge about "how to teach" the domain subject. Rather, the Teacher itself is involved in a "learning" process and it "learns" what teaching material has to be presented to the particular Student. To achieve this, we assume that the Teacher follows the Socratic model of learning by teaching the material using questions that are presented to the Students. It 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) Student 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 [114] 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 its own, and can also learn from its "colleagues" if it 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.

3.4 Overall Proposed Model

To approach the general aim of this Thesis, we shall now develop the overall Tutorial-*like* system, "piece by piece", where each module uses stochastic LA. The practical value of our approach is in the role it can play to emulate the conventional tutor in a pragmatic way. Of course, to do this, we shall use the properties of LA which can learn the solution to a problem without any *a priori* information about the optimal action. We thus have to also see what can be appropriately modeled as the Environment, using which the LA can achieve the learning.

As discussed, our aim will be to use LA in every module of the system. The system is composed of different software modules which are capable of communicating with the model for the Teacher, and with each other. It will thus consist of the following six models:

3.4.1 The Model for the Student Simulator

In our Tutorial-like system, there is no *real-life* Student. Each Student is represented by a Student Simulator, which tries to mimic his behavior and actions. This is the actual entity that has to be taught the material by the Teacher. This modeling enables the Tutorial-*like* system to function and be tested without the need for *real-life* Students. The details of this model are explained in Section 4.3. We shall see

there that the crucial features of the model involve the following facets:

- The Student Simulator is modeled using an LA paradigm. It uses LA as the learning mechanism to try to mimic the behavior and the learning of the Student.
- In addition to being able to learn from the Teacher, the Student Simulator is able to communicate with other Student Simulators, to enable it (them) to exchange information and learn from each other.
- While the Student Simulator is regarded as a black box to other modules in the Tutorial-*like* system, it can be, internally, composed into different modules, namely, the interface module, the classification module, the action module, and the learning module. This architecture makes it easier to implement, and to improve the performance of the Student Simulator.

Section 4.3 will explain each of these issues in greater detail.

3.4.2 The Model for the Student

This model is a *representation* of the behavior and status of the Student (or the Student Simulator). It guides the Tutorial-*like* system to take tailored pedagogical decisions customized to each Student or his simulator. One can think of this as the model which the *system* retains in order to represent the Student, even though the Student may be learning using a completely different philosophy. Thus, for example, while the *real-life* Student may be learning using a neural network strategy, the system may model his learning paradigm using an L_{RI} scheme.

Attempting to understand how a learning mechanism within a "black box" learns is by no means trivial. Indeed, as far as we know, this is an unsolved problem. To achieve this, we shall use the hierarchal philosophy as explained briefly below.

- The model of the Student will be inferred using a higher-level of LA, referred to as the *Meta-LA*. While the Students use the Tutorial-*like* system, the *Meta-LA* attempts to characterize their learning model. The input for the *Meta-LA* is obtained from its Environment, and by observing the input/output characteristics of the Student Simulator.

- The *Meta-LA* will try to determine if the Student in question is one of the flowing types:

  - A Fast Learner.
  - Normal-paced Learner.
  - A Slow Learner.

  Clearly, this sub-division of capabilities can be subject to a more fine resolution, but we believe that such a resolution is sufficient for most tutorial applications.

- The *Meta-LA* will determine the learning model of the Student by:

  - Observing and watching the consequent actions of the Student Simulator and his performance.
  - Examining the Environment's characteristics, as well as observing its response to the Student's actions.

- The *Meta-LA* Environment monitors the "performance" of the Student LA over a period of time. Depending on the sequence of observations, its Environment will penalize or reward the action of the *Meta-LA*.

- The higher-lever *Meta-LA* and the lower-level LA are connected as a Network of LA that have a unique interaction model. In this model, the *Meta-LA* is not directly affected by the lower-level (Student) LA, but rather the *Meta-LA*'s Environment monitors the progress of the Student LA. Such a interconnection modeling is novel to the field of LA.

Chapter 4 will explain the different components of this model in detail.

3.4.3 The Model for Student-Classroom Interaction

The Tutorial-*like* system allows the Student to be a member of a *Classroom* of students, where, for the most part, each member of the Classroom is permitted to not only learn from the Teacher(s) but also to "extract" information from any of his colleague students.

The Student-Classroom interaction is intended to maximize the learning experience of each Student as well as the collective learning of the Students. The Chapter will propose a feasible model for the real-life scenario in which each Student can be provided with multiple sources of knowledge, each of which source is uncertain or inaccurate.

When interacting with each other, a Student, or the Student Simulator, can select an interaction strategy to communicate with other Students. Such a strategy can be one of the following:

1. He always assumes that the knowledge of other Students is reliable.
2. He assesses the knowledge received from other Students. If he considers this knowledge credible, he agrees to utilize this knowledge.
3. He initially accepts to give due consideration to the knowledge from other Students. He then evaluates the knowledge received after a period of probation, and "unlearns" the information gleaned if he infers that this information was misleading.
4. He decides to learn independently, and does not communicate with other fellow Students.

On the other hand, the Student defines how he is willing to take advantage of the knowledge that his colleagues can offer. Our model proposes two approaches in which the Student can handle this knowledge:

- A complete transfer of information, in which case the Student will acknowledge all the knowledge from the other Student, and forget or erase his learned state completely.
- A partial utilization of information, where the Student will use the knowledge from the other Student only to enhance his knowledge but not to erase the knowledge he has gained otherwise.

In order to facilitate the Student-Classroom interaction, the Student, or the Student Simulator, uses what we refer to as a *Tactic-LA*, to decide about the initiation of interaction steps. The *Tactic-LA* will enable the Student to consider one of the following methods of interaction:

- He is willing to offer his assistance to other Students in the Classroom.
- He is looking for assistance from other Students.
- He is not interested in any of the above options.

Chapter 5 will explain the different components of this model in detail. In all brevity, the Chapter will show that such a Student-Classroom interaction is beneficial to most Students, especially the weaker ones, when they utilize the information they get from superior colleagues. Such a conclusion is, of course, intuitive, and we believe that our model is the pioneering one in this context.

3.4.4 The Model for the Domain

This model will encapsulate the Domain knowledge teaching material that needs to be communicated to the Students. It is the control center that encompasses the entire Domain knowledge, which the Teacher uses to impart knowledge to the Students. This module permits the Tutorial-*like* system to model and implement the Domain knowledge using a Socratic philosophy and *via* multiple-choice questions. It will

represent the increasing complexity/difficulty of the material being taught. In essence, the features of this model will be as follows:

- The Domain knowledge will be presented *via* multiple-choice Socratic-type questions. For each question, every choice has an associated probability of being correct. The choice with the highest reward probability is the answer to that question.
- The knowledge imparted is arranged in chapters, each of which will have a set of questions.
- Each chapter represents a level of complexity/difficulty that is harder than the previous one.
- Students will not be able to predict the answer for subsequent chapters using prior knowledge.

The Domain model we propose is capable of increasing the complexity of the Domain knowledge by reducing the range of the penalty probabilities of all actions (i.e., the multiple-choice answers), by incorporating a scaling factor μ. This will result in a clustering of the penalty probabilities, which will then lead to making the questions more complex inasmuch as it will be more difficult for the Student to infer the correct action.

The details of this model will be illustrated in Chapter 6.

3.4.5 The Model for the Teacher

In this module, we present a formal model for the Teacher to teach the Socratic-type Domain knowledge, *via* multiple-choice questions. This knowledge is also used to *test* the Students in the imparted knowledge. The aim of this part of the study is to illustrate how the Stochastic Teacher can not only present Socratic-type Domain

knowledge to the Students, but also model the way by which he can assist them so as to be able to learn increasingly complex material. Modeling the Teacher involves the following concepts:

- The Teacher retrieves Domain knowledge from the Domain model, so as to present it to the Student.
- The Domain knowledge, as mentioned in Section 3.4.4, is of the Socratic-type and is stored in the Domain model, via multiple-choice questions.
- The Domain knowledge is modeled with increasing complexity, which is intended to represent the increasing complexity of subsequent *chapters* that the Student encounters.
- The Students will learn the Domain knowledge from the questions presented to them, and from the feedback obtained by answering these questions.
- The Teacher possesses a formal strategy to improve the ability of the Students to learn more complex material. He does this by assisting them with additional information so as to handle the Domain knowledge as the difficulty of the latter increases.

In order for the Teacher to assist the Students to handle complex Domain knowledge, we propose that he provides them with *hints*. These hints will be imparted to the Students in the form of increasing the initial probability for one of the actions in the Student Simulator's action probability vector. The Teacher has the ability to control the probability that the Student correctly receives the *hint*.

The different components of this model will be explained in Chapter 7. In all brevity, the Chapter will demonstrate that the added assistance of the Teacher is beneficial to most Students, especially the weaker ones.

3.4.6 The Model for Improving the Teacher

Our Tutorial-*like* system allows the Teacher to "learn" and improve his "teaching skills" while he is using the system. This is accomplished by providing a higher-level LA, referred to as the *Meta-Teacher*, which will infer the required customization that the *Teacher* needs for the particular Student. The salient features of this model are as follows:

- The Teacher will utilize the Student model. As explained in Section 3.4.2, this will be done by using the *Meta-LA* to infer the learning model of the Student and his learning capabilities.
- The *Meta-Teacher*, as a higher-level learning concept, will try to infer the required customization that the *Teacher* needs for the specific Student.
- For each Environment that the Student is attempting to learn from, the Teacher will define his *own* standards for the specific Environment.
- The *Meta-Teacher* will make its inferences based on observing the progress of the Students at an intermediate stage of the learning.
- Based on the knowledge inferred from the *Meta-Teacher* and the Student model, the Teacher will be able to customize the teaching material presented, and provide the appropriate *hints* to each individual Student.
- This customization demonstrates the adaptability of the Teacher to the particular needs and skills of each Student.

The details of this this model will be explained in Chapter 8.

3.4.7 The Overall Prototype of the Tutorial-*like* System

Chapter 9 involves the overall implementation of our Tutorial-*like* system. In the Chapter, we report the way a researcher can use the system and how he can interact with it. We provide a sequence of screenshots of the prototype that we have implemented, as well as a detailed description of each module's task and functionality. In all brevity, our prototype attempts to incorporate all the features of the different models, described in Chapter 4 to Chapter 8.

Typically, a researcher who uses the system, proceeds along the following steps to test the prototype:

- Specify the configuration parameters for the Students and the Classroom. These parameters serve to define the characteristics of the Student/Classroom who interact with the Teacher. This includes the identity of each Student, the type of Student that the Student Simulator is trying to mimic (either Fast, Normal, or Below-Normal), the interaction strategy used by the Student, the rate of learning for the Student, etc.
- Define the configuration of the Domain knowledge.
- Define the learning Environment, its characteristics and how its complexity increases with the different *chapters*.
- Provide a mechanism by which the Teacher will provide *hints* to assist the Students.
- Define how the Teacher himself will improve his teaching skills and abilities.

By the end of the simulations, the researcher will be able to graphically observe the progress of each Student. How each student has learned, and the effect of the interaction between the Students will be depicted in these graphs.

The details of the overall system are provided in Chapter 9.

3.4.8 Communication Protocol

These models communicate with each other through a predefined protocol, namely, by exchanging messages between themselves. In this Thesis, we assume a simplistic protocol, and will thus not belabor this issue. Figure 3.1 shows the different components of the system and their interaction. Also, all of these concepts and individual modules will have to be incorporated into the overall prototype, as was explained in Section 3.4.7.

3.5 Conclusion

In this Chapter we have presented an overview of the proposed Tutorial-*like* system from a higher-level perspective. The sketch essentially shows the interaction of the modules with each other. This perspective presents an overall picture of the system in its entirety, and explains how each module fits within the context of the Tutorial-*like* system.

In the next few Chapters, we will discuss the design and implementation details of each of these modules, and the way in which they interact with each other.

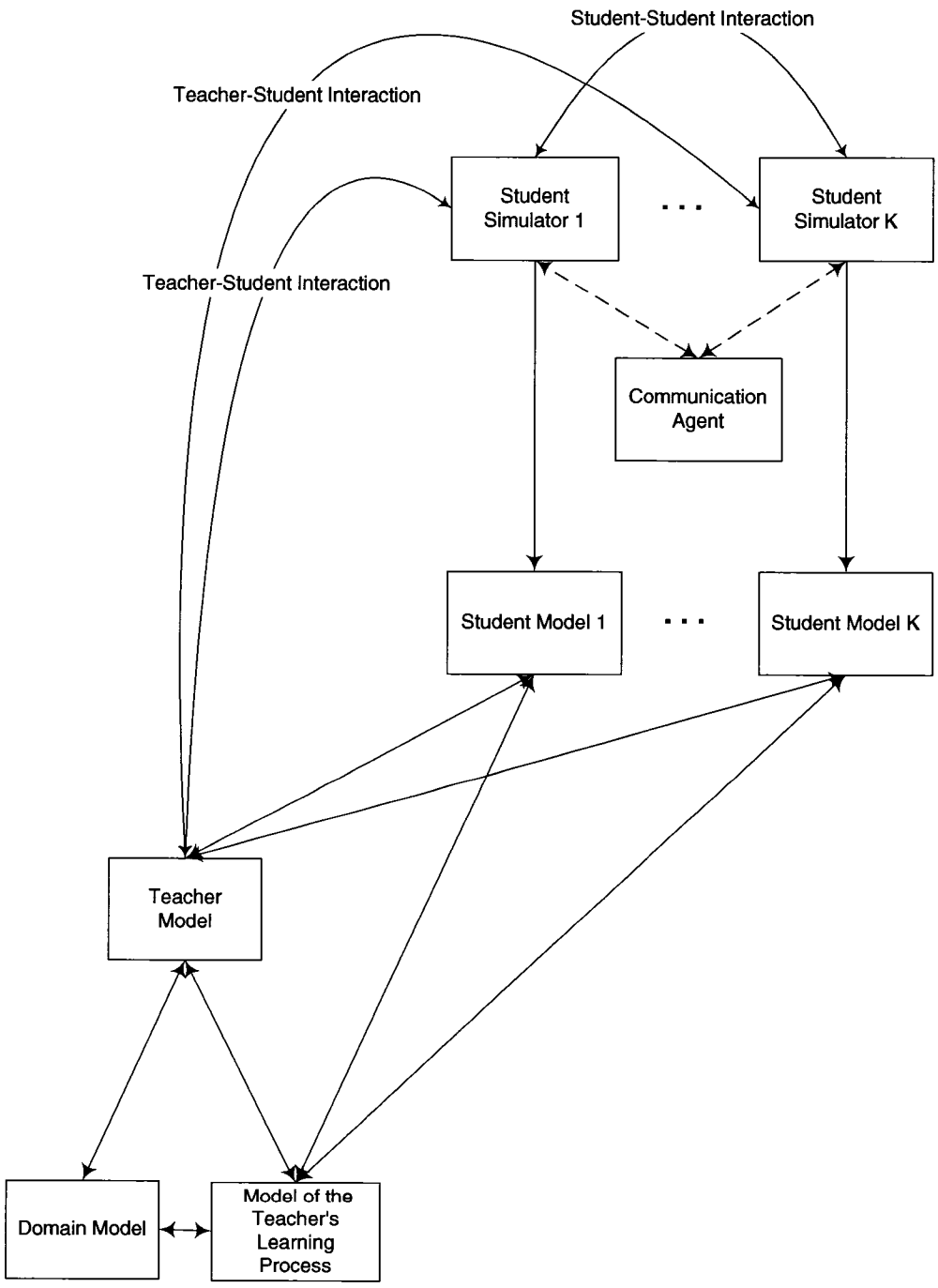

Figure 3.1: Components of the LA-based Tutorial-*like* System.

Chapter 4

Modeling the Student's Behavior

The aim of this Chapter[1] is to present a new philosophy to model the behavior of a Student in a Tutorial-*like* system using Learning Automata (LA). The model of the Student in our system is inferred using a higher-level of LA, referred to *Meta-LA*, which attempts to characterize the learning model of the Students (or Student Simulators), while the latter use the Tutorial-*like* system. The *Meta-LA*, in turn, uses LA as a learning mechanism to try to determine if the Student in question is a fast, normal, or slow learner. The ultimate long term goal of the exercise is the following: If the Tutorial-*like* system can understand how the Student perceives and processes knowledge, it will be able to customize the way by which it communicates the knowledge to the Student so as to attain an optimal teaching strategy.

This Chapter will also describe in detail the different components of the model of the Student Simulator. This model mimics the behavior of the Student while the latter uses the Tutorial-*like* system. The Student Simulator is considered as a black box to the other modules in the system to simulate their interaction with a Student. The model of the Student Simulator will be relatively brief. Notice that if we are

[1]The work done in this Chapter will be published in the *Proceedings of IEA/AIE 2007: The 20th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems*, Kyoto, Japan, June 2007 [46].

dealing with *real-life* students, the question of having a Student Simulator is non-existent. Thus, these issues are pertinent only in the context of Tutorial-*like* systems. In particular, we will discuss the various modules of the Student Simulator, namely, the interface module, the classification module, the action module, and the learning module. The rest of the Chapter will concentrate on the learning module, where the system will attempts to learn the best model used by the Student Simulator.

As we will see in the body of this Chapter, the proposed *Meta-LA* scheme has been tested for numerous environments including the established benchmarks, and the results obtained are remarkable. Indeed, to our knowledge, this is the first result which attempts to infer the learning model of an LA when it is treated externally as a black box, whose outputs are the only observable quantities. Additionally, our Chapter represents a new class of Multi-Automata systems, where the *Meta-LA* communicate with the Students, *also modeled* using LA, in a synchronous scheme of interconnecting automata. However, whereas the traditional concepts of interconnecting automata model the response of the Environment of one automaton to be the input for the other automaton, our *Meta-LA* are not affected directly by the response of the other automaton. Thus, the *Meta-LA*'s Environment "observes" the progress of the Student LA, and the response of the latter to the *Meta-LA* actions are based on these observations.

4.1 Introduction

The Student is the focal point of interest in any Tutorial system. The latter is customized to maximize the learning curve for the Student. In order for the Tutorial system to achieve this, it needs to treat every student according to his particular skills and abilities. The system's model for the Student permits such a customization.

The Student model is a representation of the Student's behavior and status. It is the basis for representing the state of the Student. When the Student uses the system, the Student model records his sequence of actions performed, using which

the system should, hopefully, model the learning paradigm used by the Student, and should attempt to assess the Student's learning progress.

The aim of this Chapter is to present a new philosophy to model how a Student learns while using a Tutorial-*like* system, that is based on the theory of stochastic Learning Automata (LA). The Stochastic LA will enable the system to determine the learning model of the Student, which will, consequently, enable the Tutorial-*like* system to improve the way it "deals" (i.e., presents material, teaches and evaluates) with the Student to customize the learning experience.

Inferring the Student's learning model, by what we call *Meta-LA*, will improve the learning experience for the Student. If the Teacher knows, in advance, what the learning model of the Student is, he will customize his teaching strategy according to this learning model. For example, if the Teacher knows that he is dealing with a smart Student, then he will present more difficult problems to the Student from the start, and be able to present material at a faster rate.

In this Chapter, we propose a *Meta-LA* strategy which can be used to examine the Student model. As the name implies, we have used LA as a learning tool/mechanism which steers the inference procedure achieved by the *Meta-LA*. To be more specific, we have assumed that a Student can be modeled as being one of the following types:

- Slow Learner Student.
- Normal Learner Student.
- Fast Learner Student.

Clearly, this sub-division of capabilities can be subject to a finer resolution, but we believe that such a resolution is sufficient for most tutorial applications.

In order for the system to determine the learning model of the Student, it monitors the consequent actions of the Student and uses the *Meta-LA* to unwrap the learning model. The proposed *Meta-LA* scheme has been tested for numerous environments

including the established benchmarks, and the results obtained are remarkable. Our experimental results, presented and explained later, show that the *Meta-LA* are successful in determining the learning model of the Student from observing the Student's actions and the teaching Environment. This would have a direct effect on the effectiveness of the Tutorial-*like* system.

Indeed, to our knowledge, this is the first reported result which attempts to infer the learning model of an LA when it is treated externally as a black box, whose outputs are the only observable quantities.

4.2 Contributions of this Chapter

This Chapter presents a novel approach for modeling how the Student learns in a tutoring session. We believe that successful modeling of Students will be fundamental in implementing the task of effective Tutorial-*like* systems. Thus, the salient contributions of this Chapter are:

- The concept of *Meta-LA*, in Student modeling, which helps to build a higher-level learning concept, that has been shown to be advantageous for customizing the learning experience of the Student in an "efficient" way.
- The Student Modeler, using the *Meta-LA*, will observe the action of the Student concerned and the characteristics of the Environment. It will use these observations to build a learning model for each Student. This will enable the Student Modeler to find out if the Student is a slow, normal, or fast learner.
- While a traditional Tutorial system may not be able to consider the "psychological" perspective of the Students, we attempt to also use a model of their cognitive state. After the Tutorial-*like* system attempts to infer the correct learning model of the Student, it will be able to customize its teaching strategy as per the Student's learning ability.

4.3 Student Simulator Model

The fundamental premise of our study is to, if necessary, eliminate the need for *real-life* students using our Tutorial-*like* system. Thus, effectively, the system could be used by either:

- A Student Simulator that mimics the Student behavior.
- An artificial entity which, in itself, could be another software component that needs to "learn" specific Domain knowledge from a field.

The Tutorial-*like* system uses a representation of the Student, which is the Student Simulator. This will enable the Tutorial-*like* system to function and to be tested without the need to involve *human* (i.e., real-life) entities – which is a logistically complex problem in itself. One of the disadvantages of any real-life system with real-life students (including those involving "intelligence") is that it has to be tested by many students before it can reach the optimal teaching strategy. Of course, the numerous psychological and statistical testing issues have also to be addressed in a real-life scenario. Using a Student Simulator model will, hopefully, remedy this disadvantage, and will enable the Tutorial-*like* system to reach the optimal strategy without requiring human "guinea pigs" to interact with a *sub-optimal* tutorial system.

Simulating the Student is far from trivial. Here too, we propose that the Student Simulator model uses stochastic LA as a learning mechanism to imitate how a Student would behave, and to learn in a tutoring session. Our hypothesis is that using LA would allow the simulator model to demonstrate the characteristics of a real-life student, and as the Student's interaction with the tutorial-*like* system increases, the learning of the LA would hopefully also increase, implying that the simulator model would better represent the learning of a real-life student. It is our vision that after the system is perfected, the Student Simulator model can be "replaced" by real-life students who can, in turn, use a "perfected" tutorial (not tutorial-*like*) system.

In the following section, we will describe the different components of the model of the Student Simulator, which is composed into different modules; the interface module, the classification module, the action module, and the learning module. This will provide an overview of the design of the model of the Student Simulator, and how it functions internally, while it is viewed as a black box to the other modules in the system. After that, the Chapter will focus on the learning of the model of the Student Simulator, where the system will try to determine the best learning model used by the Student Simulator.

4.3.1 Components of the Simulator Model

The Student Simulator model will be composed of different components. Figure 4.1 shows these different components and the interaction between them. Furthermore, these components are laid out in modules as explained below.

4.3.1.1 Interface Module

The interface module includes the communication interface, which provides a common gateway between the Student Simulator and the other models listed in Section 3.4. This communication with other models in the systems, in turn, is achieved through a common protocol provided by the Student Simulator. The protocol allows the Student(s) to receive requests from other models, and to also initiate requests to other models in the system.

4.3.1.2 Classification Module

In this module the Student Simulator distinguishes between the requests from the various sources. It differentiates between interacting with the Teacher, and interacting with fellow Students. Students, typically, listen to teachers, take their advice with

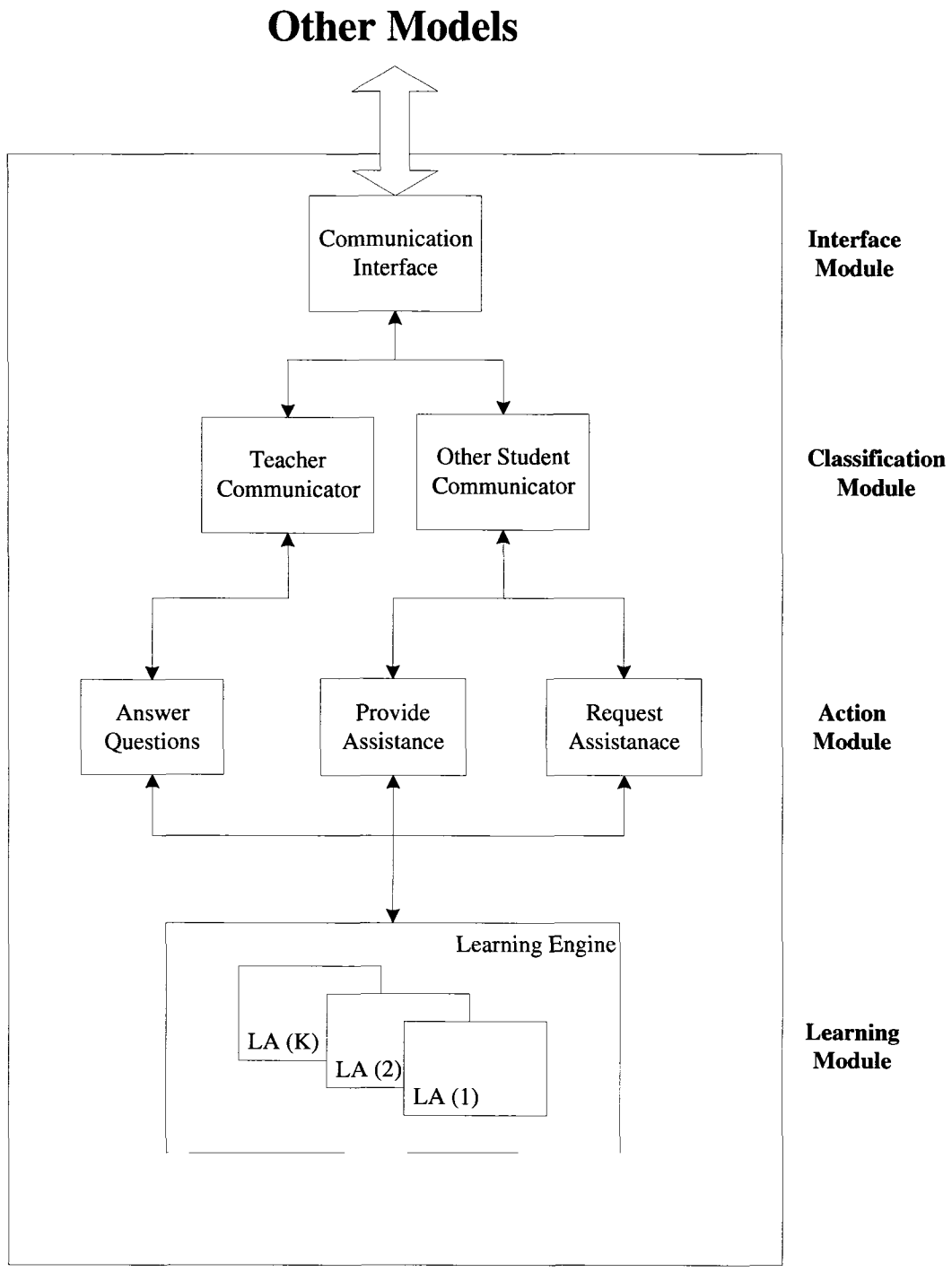

Figure 4.1: Components of the Student Simulator Model.

confidence, and respond promptly to any questions they are asked. However, interaction with another Student may be categorized otherwise. Indeed, Students typically give less confidence to information obtained from another students, and they may also be reluctant to query another Student. Finally, the Student may decide to postpone or cancel an interaction with another Student.

4.3.1.3 Action Module

The Student Simulator, while interacting with other models, can embark on either a reactive or proactive step. A reactive step is a response for a request from another model, for example one which is encountered when answering a question from the Teacher. The Student Simulator can also take proactive steps when interacting with the teaching environment, as described by one of the following phenomena:

1. When it interacts with another Student.
2. When it proposes to the system, that it is willing to help another Student.

In the action module, the Student Simulator executes the required action, and generates the required response for that specified action. An action could be any one of a set of possible tasks such as answering questions, requesting help from a colleague Student, or providing assistance to other Students.

4.3.1.4 Learning Module

The Student Simulator, hopefully, learns and improves its behavior with time. It uses LA as an internal learning mechanism to learn from its previous actions, and receives feedback from the pertinent environment, in question, to improve the decisions it takes in the future.

The LA used by the Student Simulator could be different for each Student. Our

task will be to consider how different types of LA could be used to model different Students.

LA are also used to simulate the *individual* proactive decisions of the Student Simulator. Later, in Chapter 4, the *Tactic-LA* will be introduced as a higher-level supervisory module of the Student Simulator's LA, which can be used to simulate the decision process of the Students to decide whether to interact with other colleagues in the Classroom or not.

4.4 Network Involving the LA/*Meta-LA*

Our Tutorial-*like* system incorporates multiple LA which are, indirectly, interconnected with each other. This can be perceived to be a system of interconnected automata [141]. Since such systems of interconnecting automata may be one of two models, namely, synchronous and sequential, we observe that our system represents a synchronous model. The LA in our system and their interconnections are illustrated in Figure 4.2.

In a traditional interconnected automata, the response from the Environment associated with one automaton represents the input to another automaton. However, our system has a rather unique LA interaction model. While the Student LA is affecting the *Meta-LA*, there is no direct connection between both of them. The *Meta-LA* Environment monitors the "performance" of the Student LA over a period of time, and depending on that, *its* Environment would penalize or reward the action of the *Meta-LA*.

This model represents a new structure of interconnection which can be viewed as being composed of two levels: a higher-level automaton, i.e., the *Meta-LA*, and a lower-level automaton, which is the Student LA. The convergence of the higher-level automaton is dependent on the behavior of lower-level automaton. A typical scenario would be that the higher-level automaton (which infers the type of learning achieved

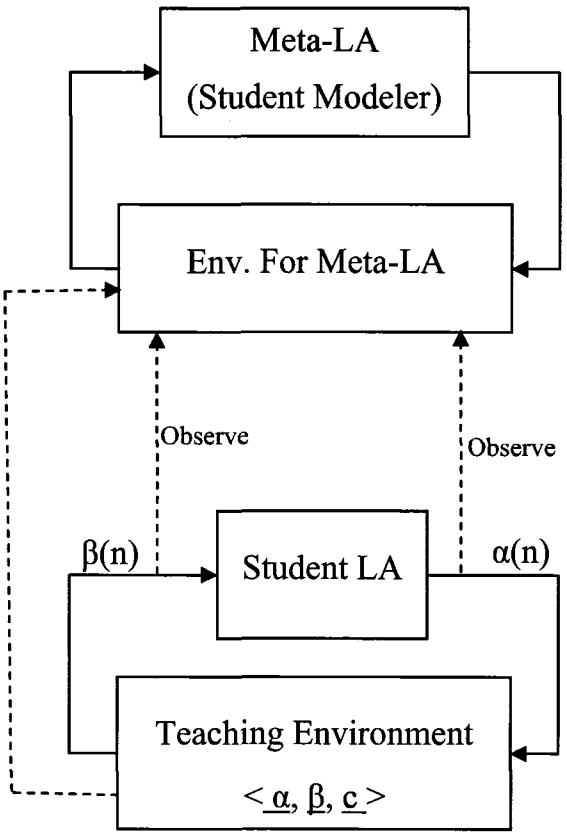

Figure 4.2: Modeling Using a Network of LA.

by the lower-level LA) converges, while the lower-level automaton has yet to converge. If the lower-level automaton converges before the higher-level automaton, this could mean that the convergence of the higher-level automaton is inaccurate.

Observe the uniqueness of such an interaction of modules, as a consequence of the following:

1. The *Meta-LA* has no Environment of its own.
2. The Environment for the *Meta-LA* consists of observations, which must be perceived as an Environment after doing some processing of the signals observed.
3. The *Meta-LA* has access to the Environment of the lower level, including the set of the penalty probabilities that are unknown to the lower-level LA.

4. The *Meta-LA* has access to the actions chosen by the lower-level LA.

5. Finally, the *Meta-LA* has access to the responses made by the lower-level Environment.

In conclusion, we observe that items (3)-(5) above, collectively constitute the Environment for the *Meta-LA*. Such a modeling is unknown in the field of LA.

4.5 Modeling a Student

The Tutorial-*like* system derives a model for the Student by assessing and determining the way he learns. To achieve this, we assume that the system has a finite set of possible learning models for each Student. For example, each Student can be modeled as being one of the following types:

1. A Fixed Structure Automaton (FSSA) model, which represents a Slow Learner.

2. A Variable Structure Automaton (VSSA) model, which represents a Normal Learner.

3. A Pursuit model LA, which represents a Fast Learner.

This finite set of learning models represents the different families that characterize the way the Student learns. The Student could be, for example, a slow learner, a normal leaner, or a fast learner. Of course, a finer learning categorization is also achievable. As mentioned, if the Tutorial-*like* system can understand how the Student perceives knowledge, it will be able to customize the way by which it communicates the knowledge to the Student so as to aim towards an optimal teaching strategy. For example, if the Tutorial-*like* system finds that the Student is learning in a way that resembles a Pursuit model, it can conclude that the Student is "clever", and that it is a

fast learner. In this case, the system can adapt its teaching strategy to accommodate a fast-learning Student, which, in turn, enables it to improve its teaching strategy.

The Student Modeler will itself utilize LA in the *Meta-LA* level in order to represent the different potential learning models for the Student. Also, as the Student interaction with the Tutorial-*like* system increases, *this* LA would, hopefully, converge to the model that most accurately represents the way the Student learns.

4.5.1 Formalization of the Student Model

To simplify issues, we assume that the learning model of the Student is one of the three types mentioned above[2]. The Student model will use three specific potential LA, each of which serve as an action for a higher-level automaton – (i.e., a *Meta-LA*).

In our present instantiation, the *Meta-LA* which is used to learn the best possible model for the Student Simulator is a 4-tuple: $\{\underline{\alpha}, \underline{\beta}, \mathrm{P}, T\}$, where:

$\underline{\alpha} = \{\alpha_1, \alpha_2, \alpha_3\}$, in which

α_1 is the action corresponding to a FSSA model, which represents a Slow Learner.

α_2 is the action corresponding to a VSSA model, which represents a Normal Learner.

α_3 is the action corresponding to a Pursuit model LA, which represents a Fast Learner.

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

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

[2]Extending this to consider a wider spectrum of learning models is rather straightforward.

and

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

$P = [p_1, p_2, p_3]^T$, where

$p_i(n)$ is the current probability of the Student Simulator being represented by α_i.

T is the probability updating rule given as a map as:

T: $(P, \underline{\alpha}, \underline{\beta}) \rightarrow P$.

Each of these will now be clarified.

1. $\underline{\beta}$ is the input which the *Meta-LA* receives (or rather, infers). Let us suppose that the current choice of the *Meta-LA* is that the Student Simulator is learning using the model symbolized by α_i. This means that the *Meta-LA* will have to infer a Reward or a Penalty based on how the Student Simulator is learning, and on whether the latter can be accurately represented by α_i or not. This, in turn, means that the *Meta-LA* must observe a sequence of decisions made by the Student Simulator, and based on this sequence it must deduce whether its current model is accurate or not. We propose to achieve this as follows.

   For a fixed number of queries, which we shall refer to as the "Window", the *Meta-LA* assumes that the Student Simulator's model is α_i. Let $\theta(t)$ be the rate of learning of the lower LA at time 't'. By inspecting the way by which the Student Simulator learns within this Window, it infers whether this current model should be rewarded or penalized. We propose to achieve this by using two thresholds, which we refer to as $\{\theta_j | 1 \leq j \leq 2\}$ (as shown in Table 4.1). The significance of these thresholds is the following:

   - If the current observed learning rate of the Student Simulator is less than θ_1, we assert that the Student Simulator demonstrates the phenomenon

Type Of Student	Boundary Threshold
Slow Learner	$\theta(t) < \theta_1$
Normal Learer	$\theta_1 \leq \theta(t) < \theta_2$
Fast Learer	$\theta_2 \leq \theta(t)$

Table 4.1: Thresholds used by the *Meta-LA* for the Learning Model, where $\theta(t)$ is the Rate of Learning at time 't'.

of a *Slow* Learner. Now, if the *Meta-LA* has chosen α_1, this choice is rewarded. Otherwise, this choice is penalized.

- If the current observed learning rate of the Student Simulator is equal or greater than θ_1 and less than θ_2, we assert that the Student Simulator demonstrates the phenomenon of a *Normal* Learner. Thus, if the *Meta-LA* has chosen α_2, this choice is rewarded. Otherwise, this choice is penalized.
- Finally, if the current observed learning rate of the Student Simulator is equal or greater than θ_2, we assert that the Student Simulator demonstrates the learning properties of a *Fast* Learner. Thus, if the *Meta-LA* has chosen α_3, this choice is rewarded. Otherwise, the *Meta-LA* is penalized.

This terminates our discussion on how β is inferred.

2. P is the action probability vector which contains the probabilities that the *Meta-LA* assigns to each of *its* actions. At each instant t, the *Meta-LA* randomly selects an action $\alpha(t) = \alpha_i$. The probability that the automaton selects action α_i, at time t, is the action probability $p_i(t) = \Pr[\alpha(t) = \alpha_i]$, where $\sum_{i=1}^{r} p_i(t) = 1 \quad \forall\, t$, i=1, 2, 3.

   Initially, the *Meta-LA* will have an equal probability for each of the three learning models as:

   $$[0.333 \quad 0.333 \quad 0.333]^{\mathrm{T}}$$

By observing the sequence of $\{\theta(t)\}$, the *Meta-LA* updates $P(t)$ as per the function T, described presently. By virtue of this interaction, one of the action probabilities, p_j, will, hopefully, converges to a value as close to unity as we want. This convergence indicates that the *Meta-LA* has converged to the choice that α_j is the best learning model, and if T is chosen appropriately, we believe that it will be successful in determining the best learning model appropriate for the Student Simulator.

3. T is the updating algorithm, or the reinforcement scheme used to update the *Meta-LA*'s action probability vector, and is represented by:

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

   This updating algorithm could follow *any* LA learning schemes. For example, if it obeys the DL_{RI} rule, the action probability vector is updated in a *discretized* manner when the *Meta-LA* is rewarded, and is unchanged when the automaton is penalized.

4.6 Experimental Results

This section presents the experimental results obtained by testing the prototype implementation of the *Meta-LA*. In order to obtain these results we performed numerous simulations to accurately simulate how the Student Modeler is able to recognize the learning model of the Student.

The simulations were performed for two different types of Environments, two 4-action Environments and two benchmark 10-action Environments. These Environments represent the teaching "problem", which contains the uncertain course material of the Domain model associated with the Tutorial-*like* system. In all the tests performed, an algorithm was considered to have converged, if the probability of choosing an action was greater than or equal to a threshold $T(0 < T \leq 1)$. If the automaton converged to the best action (i.e., the one with the highest probability of being

rewarded), it was considered to have converged correctly.

As presented in Section 4.4, the *Meta-LA* was implemented as a higher-level LA, which had access to the learning Environment and the actions and penalties of the lower-level LA associated with the Student Simulators, while the latter learned the teaching material. The *Meta-LA* has been implemented as a Discretized LA, the DL_{RI} scheme, with a resolution parameter N=40.

The Student Simulator was implemented to mimic three typical types of Students as follows:

- Slow Learner: For this type of Students, the Student Simulator used a FSSA to mimic the Student's behavior. In particular, it used a Tsetlin $L_{N,2N}$ automaton implementation for this behavior, where it had two states for each action.
- Normal Learner: For this category of Students, the Student Simulator used a VSSA to simulate the Student's behavior. In particular, in our study, it was implemented as an L_{RI}, with λ=0.005. Observe that, the lower-level LA updates itself only if it obtained a reward for its action; otherwise, it did nothing.
- Fast Learner: To simulate Students of this type, the Student Simulator utilized a Pursuit LA to represent the Student's behavior. It used the Pusuit PL_{RI}, with λ=0.005, to simulate fast learning Students. Here too, the LA only updated its probabilities when it obtained a reward for its action.

The rate of learning associated with the Student is a quantity that is difficult to quantify. We had to determine the best index in each case. To be able to measure θ, the rate of learning, for the Student, the *Meta-LA* Environment calculates the loss-function of the Student Simulator LA. For a "Window" of n choice-response feedback cycles of the Student Simulator, the loss-function is defined by:

$\frac{1}{n} \sum n_i \, c_i$, where,

n_i is the number of instances an action α_i is selected inside the "Window"

and,

c_i is the penalty probability associated with action α_i of the lower-level LA.

We consider the rate of learning for the Student to be a function of the loss-function and time, as illustrated in Figure 4.3. This reflects the fact that the loss function for any learning Student, typically, decreases with time (for all types of learners) as he learns more, and as his knowledge improves by time.

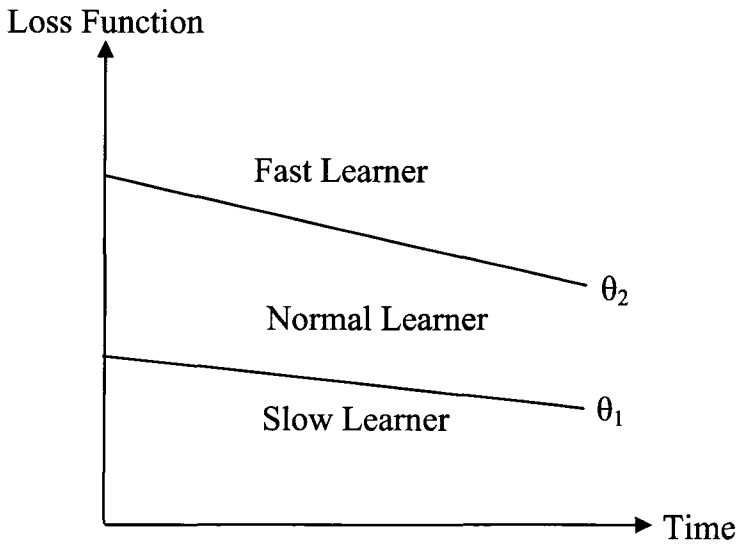

Figure 4.3: Loss Function and θ.

As mentioned, the simulation has been performed for the different existing benchmark Environments with 4 actions and 10 actions, 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.

4.6.1 Environment with 4-Actions

The 4-action Environment represents a Multiple-Choice question with 4 options. The Student needs to learn the content of this question, and conclude that the answer for this question is the action which possesses the minimum penalty probability. Based

Env.	θ_1	θ_2	PL_{RI}		L_{RI}		Tsetlin	
			No. of Iter.	% correct converg.	No. of Iter.	% correct converg.	No. of Iter.	% correct converg.
$\mathbf{E_A}$	0.50-0.31	0.60-0.35	607	100%	1037	87%	1273	91%
$\mathbf{E_B}$	0.28-0.17	0.37-0.24	790	97%	1306	89%	1489	85%
Reward probabilities are: $\mathbf{E_A}$: 0.7 0.5 0.3 0.2 $\mathbf{E_B}$: 0.1 0.45 0.84 0.76								

Table 4.2: Convergence of *Meta-LA* for Different Student's Learning Models in 4-Action Environments.

on the performance of the Student, the *Meta-LA* is supposed to "learn" the learning model of the Student. The results of this simulation are presented in Table 4.2.

To be conservative, the simulation assumes that when the Student starts using the system, the results during a transient learning phase, are misleading and inaccurate. Consequently, the first 200 iterations are neglected by the *Meta-LA*, where it merely ignores all the interactions, decisions, and responses of the lower-level LA.

The two different settings for the two Environments are given by their reward probabilities:

$$E_A = \{0.7 \quad 0.5 \quad 0.3 \quad 0.2\} \text{ and,}$$

$$E_B = \{0.1 \quad 0.45 \quad 0.84 \quad 0.76\}.$$

The results obtained were quite remarkable. Consider the case of E_A, where the best action was α_1. While an FSSA converged in 1273 iterations[3], the L_{RI} converged in 1037 iterations, and the Pursuit PL_{RI} converged in 607 iterations. By using the

[3]The figures include the 200 iterations involved in the transient phase.

Env.	θ_1	θ_2	PL_{RI}		L_{RI}		Tsetlin	
			No. of Iter.	% correct converg.	No. of Iter.	% correct converg.	No. of Iter.	% correct converg.
E_A	0.50-0.31	0.60-0.35	668	95%	748	93%	1369	96%
E_B	0.41-0.17	0.51-0.26	616	100%	970	85%	1140	96%
Reward probabilities are:								
E_A :	0.7	0.5	0.3	0.2	0.4	0.5	0.4	0.3	0.5	0.2
E_B :	0.1	0.45	0.84	0.76	0.2	0.4	0.6	0.7	0.5	0.3

Table 4.3: Convergence of *Meta-LA* for Different Student's Learning Models in 10-Action Environments.

values of θ_1 and θ_2 as specified in Table 4.2, the *Meta-LA* was able to characterize the true learning capabilities of the Student 100% of the time when the lower-level LA used a Pursuit PL_{RI}. The lowest accuracy of the *Meta-LA* for Environment E_A was 87%. The analogous results for E_B were 97% (at its best accuracy) and 85% for the Slow Learner. The power of our scheme should be obvious !

4.6.2 Environment with 10-Actions

The 10-action Environment represents a Multiple-Choice question with 10 options. Here, again, the Student needs to learn the content of this question, and conclude that the answer for this question is the action which possesses the minimum penalty probability. The *Meta-LA* is supposed to "learn" the Student learning model by observing the performance of the Student. Table 4.3 shows the results of this simulation. Observe that the first 200 iterations are neglected by the *Meta-LA*, and thus, in the transient phase, it completely ignores all the interactions, decisions and responses of the lower-level LA.

The two Environments are characterized by their reward probabilities being:

$$E_A = \{0.7 \quad 0.5 \quad 0.3 \quad 0.2 \quad 0.4 \quad 0.5 \quad 0.4 \quad 0.3 \quad 0.5 \quad 0.2\} \text{ and}$$

$$E_B = \{0.1 \quad 0.45 \quad 0.84 \quad 0.76 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.7 \quad 0.5 \quad 0.3\}.$$

Similar to the 4-action Environments, the results obtained for the 10-action Environments were very fascinating. For example, we consider the case of E_A, where the best action was α_1. While an FSSA converged in 1369 iterations, the L_{RI} converged in 748 iterations, and the Pursuit PL_{RI} converged in 668 iterations. By using the values of θ_1 and θ_2 as specified in Table 4.3, the *Meta-LA* was able to determine the true learning capabilities of the Student 95% of the time, when the lower-level LA used a Pursuit PL_{RI}. The lowest accuracy of the *Meta-LA* for Environment E_A was 93%. By comparison, for E_B, the best accuracy was 100% for the Fast Leaner and the worst accuracy was 85% for the Slow Learner.

We believe that the worst-case accuracy of 15% (for the inexact inference) is within a reasonable limit. Indeed, in a typical learning environment, the Teacher may, incorrectly, assume the wrong learning model for the Student. Thus, we believe that our *Meta-LA* learning paradigm is remarkable, considering that these are the first reported results for inferring the nature of an LA if it is modeled as a black box.

4.7 Conclusion

This Chapter presented a novel strategy for a Tutorial-*like* system inferring the model of a Student (Student Simulator). The Student Modeler itself uses a LA as an internal mechanism to determine the learning model of the Student, so that it could be used in the Tutorial-*like* system to customize the learning experience for each Student. To achieve this, the Student Modeler uses a higher-level automaton, a so-called *Meta-LA*, which observes the Student Simulator LA's actions and the teaching Environment. It does this while attempting to characterize the learning model of the Student (or

Student Simulator), while the latter uses the Tutorial-*like* system. The *Meta-LA* tries to determine if the Student in question is a fast, normal, or slow learner.

From a conceptual perspective, the Chapter presents a new approach for using interconnecting automata, where the behaviorial sequence of one automaton affects the other automaton. The numerous other differences of such an interconnection are also listed in the Chapter.

From the simulation results, we can conclude that this approach is a feasible and valid mechanism applicable for implementing a Student modeling process. The results shows that the Student Modeler was successful in determining the Student learning model in a high percentage of the cases – sometimes with an accuracy of 100%.

In the next Chapter, we will present the Student-Classroom interaction model, in which the Student can learn not only from the Teacher, but also from a colleague Student. We will demonstrate how this interaction can be beneficial to improve the learning of the Students.

Chapter 5

Modeling Student-Classroom Interaction

Almost all of the learning paradigms used in machine learning, Learning Automata (LA), and learning theory, in general, use the philosophy of a Student (learning mechanism) attempting to learn from a Teacher. This paradigm has been generalized in a myriad of ways including the scenario when there are multiple Teachers or a hierarchy of mechanisms which collectively achieve the learning.

In this Chapter, we consider a departure from this paradigm by allowing the Student to be a member of a *Classroom* of students, where, for the most part, we permit each member of the Classroom to not only learn from the Teacher(s) but also to "extract" information from any of his colleague students. This Chapter deals with the issues concerning the modeling, decision making process and testing of such a scenario within the LA context. The main result that we show is that a weak learner can actually benefit from this capability of utilizing the information that he gets from a superior colleague – if this information transfer is done appropriately.

As far as we know, the whole concept of Students learning from both a Teacher and from a Classroom of Students is novel, and we believe that our results are of

a pioneering sort. The proposed Student-Classroom interaction has been tested for numerous strategies and for different environments including the established benchmarks. Also, the results show that Students can improve their learning by interacting with each other. For example, for some interaction strategies, a weak Student can improve his learning by up to 73% when interacting with a Classroom of Students, which includes Students of various capabilities. In these interactions, the Student does not have *a priori* knowledge of the identity or characteristics of the Students who offer their assistance.

5.1 Introduction

Interaction between Students is a source of learning for Students in real-life teaching environments. While Students usually consider the Teacher as the main source of information, typically, they also depend on each other to supplement their learning. Traditional Tutorial systems, typically, allow Students to interact only with the Teacher. This prevents the Students from gaining knowledge from a valuable source of information, which is their mutual interaction.

The aim of this Chapter is to present a new philosophy to model Student-Student interaction in a Tutorial-*like* system that is based on stochastic Learning Automata (LA). In the new model, a Student is a member of a Classroom of students, where he is not only learning from the Teacher but also obtaining information from other colleague students. In our system, a Student Simulator is used to mimic the behavior of *real-life* students during the learning process. Any interaction between the Students is represented by the interaction between different LA, and this is intended to augment the interaction of each Student Simulator with the Teacher (Environment). The goal of the exercise is to maximize the learning of each Student and the collective learning of the Students, by allowing them to also interact with each other. Therefore, we propose a feasible model for the real-life scenario in which a system can offer multiple

sources of knowledge to each Student, each of which source is uncertain or inaccurate.

In this Chapter, we propose that a Student can successfully interact and gain knowledge from other Students. He can pick an interaction strategy to communicate with other Students, and such a strategy can be one of the following:

1. He always considers that the knowledge of other Students is credible.
2. He evaluates the knowledge received from other Students. He then accepts this knowledge only if he can infer that this information is credible.
3. He evaluates the knowledge received from other Students, and after a period of probation "unlearns" the information gleaned if he infers that this information was misleading.
4. He chooses not to communicate with other fellow Students.

The Student, in turn, needs to decide as to what extent he is willing to take advantage of the knowledge that his colleagues can offer. In this Chapter, we consider essentially, two strategies by which the Student can use this knowledge:

- Complete transfer of information. In this strategy, the Student will consider all the knowledge from the other Student and forget his learned state completely.
- Partial utilization of information. In this case, the Student will use the knowledge from the other Student only to "improve" his knowledge.

In terms of protocol, interaction between the Students is facilitated by the use of a Communication Agent. Students who are interested in assisting other Students would need to register their intent with the Communication Agent. The same also applies for Students who are looking for help from other Students. The Agent will be responsible for matching these Students together.

The Student, or the Student Simulator, uses what we call a *Tactic-LA* to decide whether he should initiate an interaction with another Student or not, and this can lead to one of the following actions:

- The Student is willing to help other Students in the Classroom.
- The Student is looking for help from other Students.
- The Student is interested in neither of the above options.

The proposed Student-Classroom interaction has been tested for different interaction strategies and for numerous environments including the established benchmarks. The results obtained are excellent. Our experimental results, presented and explained later, show that our proposed model of Student-Classroom interaction can improve the learning for the Student(s). A weak learner, for example, can improve his learning by interacting with superior learners, and this can translate to improve the speed of learning for the weak learner by as much as 73% in some cases.

Indeed, to our knowledge, this is the first published result which attempts to simulate the interaction between different Students (modeled as LA) while they are simultaneously learning from the Teacher (Environment). In this novel interaction, a Student can learn from the Teacher, as well as from his colleagues in the Classroom to improve his learning process.

5.1.1 Contributions of this Chapter

This Chapter presents a novel approach for modeling how a Student can learn and interact, in a Tutorial-*like* system, not only from a Teacher, but also from a Classroom of Students. We believe that the Student-Classroom interaction is an important feature in implementing Tutorial-*like* systems, which can be utilized efficiently to improve the learning capabilities of the Students. Thus, the significant contributions of this Chapter are:

- The concept of inter-Student interactions where they are modeled by LA, while they simultaneously learn from the Teacher (Environment). This enables each Student entity found in the Tutorial-*like* system (and them collectively) to effectively improve their learning, by enabling them to "extract" information and learn from each other, in addition to the traditional model of merely learning from the Teacher.

- The introduction of the potential of enabling different strategies in the Student-Classroom interaction, where these are based on exchanging complete or partial knowledge between the participating entities.

- The concept of a *Tactic-LA* in Student Simulation, which helps the various Students to determine the actions needed to facilitate the interaction between themselves.

5.2 Network of Interacting Students

In our proposed solution, a Tutorial-*like* system incorporates multiple Students (modeled as LA) who are interacting with the Teacher (Environment), as well as interacting with each other. The problem we encounter is one of determining how we can model and implement the phenomenon of Students learning from the Teacher, and also providing them with the ability to learn by exchanging knowledge between each other. Also, open to this study, is the issue of how this *knowledge* can be represented and utilized.

We propose a novel class of multiautomata systems in which different LA are interacting to improve their learning. Unlike a game of automata, where each player is trying to find the optimal action to maximize his payoff from the environment, our model generalizes the potential interconnection. It provides for a collaborative set of players, where each player is willing to cooperate with the others in the system so as to enable the entire group to maximize their learning. However, unlike the field of

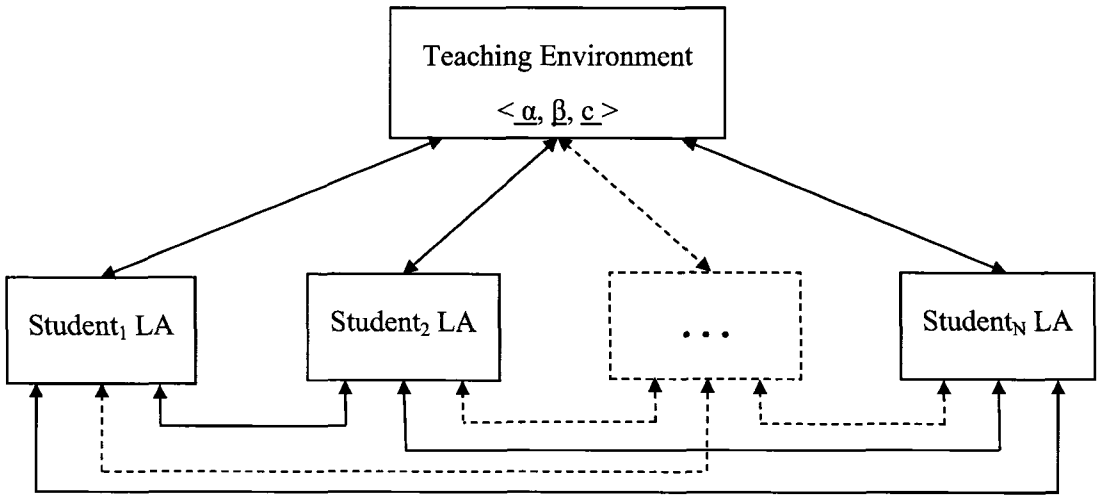

Figure 5.1: Student-Classroom Interaction Model.

cooperative games, we also permit the players to opt to play, or to remain isolated. They are, additionally, given the option to decide *how* to utilize the information provided by their colleagues.

Of course, such a generalization comes with a price, especially as we model the "quality" of the shared knowledge to be poor – the information communicated is uncertain or inaccurate. Students need to follow a strategy by which they would decide if they want to make use of the exchanged knowledge. To be more specific, in all brevity, the Students' LA in our system and their interactions with each other are illustrated in Figure 5.1.

Whenever a Student receives help from another Student, along with this knowledge he is provided with the option of using it in one of two ways:

- Complete transfer of knowledge. The Student who receives help completely replaces his knowledge with the knowledge provided by his colleague Student. Literally, he "wipes his slate clean" from his prior knowledge, and replaces it with the knowledge that is provided. This transfer of knowledge and information is facilitated in our solution by replacing the current action probability vector (for the teaching material being taught) by the one that is provided.

- Partial utilization of knowledge. In this scenario, the Student uses the knowledge of the Helper Student to *improve* his own knowledge. This, in turn, translates into the operation that the action probability vector of the Student will be affected by the transfer of the knowledge (information). Such a transfer of knowledge will lead to an incremental increase of the Student's existing knowledge. The issue of how this is achieved is described later.

The Student who receives assistance needs to select an interaction strategy to be able to decide how to handle the knowledge exchanged. In our solution, we propose that the different strategies that a Student can pursue are the following:

- **Always Consider the Other Student's Knowledge as Reliable**. In this strategy, the Student who receives help *always* assumes that the Student who offers help has a better knowledge of the Domain. This strategy is beneficial for a weaker Student if he is getting help from a superior Student. But, in practice, if the setting is reversed, it can lead to a negative effect on the Student's state of knowledge. The handling of this interaction can be achieved by utilizing either a complete or a partial exchange of the knowledge communicated *via* the action probability vector.

- **Incorporate the Other Student's Knowledge *only* if it is Reliable**. In this strategy, the Student does not blindly incorporate the knowledge provided by the Helper Student. Rather, before accepting help from any of his colleagues, he evaluates the knowledge that is imparted. If he "considers" that the knowledge provided is superior to what he has, he can choose to accept it. Otherwise, he opts to keep his knowledge intact. These decisions are enforced, again, *via* the action probability vector.

  With regard to implementation, the Student considers the knowledge offered to be superior than his if the action probability vector of the Helper Student seems to lead to a convergence better than his own. Again, the knowledge transfer can be either complete or partial.

- **"Unlearn" if the Student's Knowledge does not Improve.** In this scenario, the Student will initially accept the knowledge from the Helper Student. He will, thereafter, evaluate his own progress during a probationary period. If he concludes that his knowledge did not improve during this probationary period, he "unlearns" by discarding the information that he absorbed, and reverts back to his original knowledge.

  The evaluation of the knowledge gained from the Helper Student is done by checking if the Student's action probability vector converges towards the unit vector representing the best action – as dictated by the exchanged knowledge.

- **Transfer No knowledge.** In this strategy, the Student decides to *not* communicate with the other Students in the Classroom. Thus, he will only learn from the Teacher !!

5.3 Communication Agent

The Communication Agent is the component of the Tutorial-*like* system that is assigned to facilitate the interaction between Students. It enables the communication between Students by matching those Students who are willing to offer assistance, with their counterparts who request assistance. The *modus operandus* of the Communication Agent is illustrated in Figure 5.2 and can be characterized by the following:

- A Student registers with the Communication Agent stating that he is willing to help other Students. The Communication Agent will then add this Student to the list of Students who are willing to help others.

- A Student expresses his interest in receiving help by registering his intent with the Communication Agent. This Student's intention is then recorded by augmenting him to the list of those who need assistance.

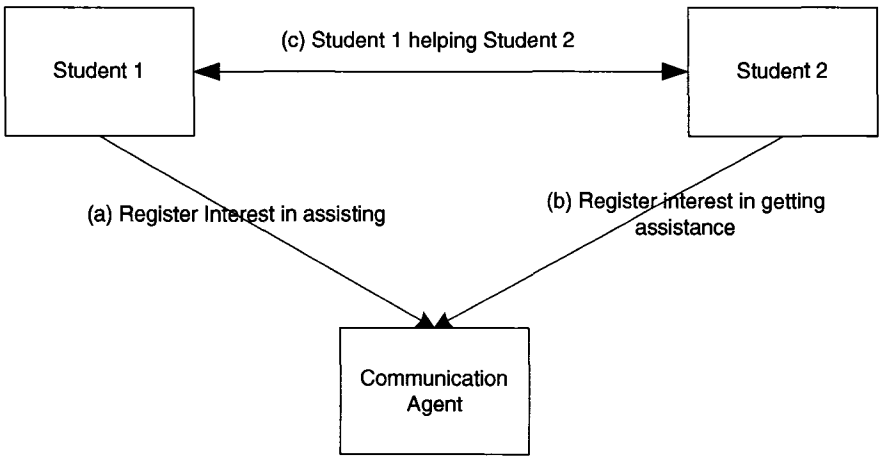

Figure 5.2: Students Interactions through Communication Agent.

- Finally, the Communication Agent will match pairs of Students together. It will then remove the Student who offers his assistance from the list of Students who offer help, and the Student who needs assistance from the corresponding list.

5.3.1 Student Simulator's *Tactic-LA*

The Student Simulator takes decisions that, hopefully, mimics a *real-life* Student. Some of these decisions can be reactive decisions, which are responses to requests from other entities in the Tutorial-*like* system. Other decisions are proactive, which the Student takes as strategic actions. One set of such proactive decisions are the ones that the Student takes to decide on how to behave in a Classroom interaction with other colleague Students.

In our model, the Student Simulator uses a so-called *Tactic-LA*, which dictates the decision making process of the Student, so as to efficiently and proactively communicate with his colleague Students in the Classroom. As the name implies, it uses LA in its own right as a learning mechanism to make these decisions. The *Tactic-LA* decides about the interaction method that the Student should take, which can be one

of three possible actions:

- The Student is interested in assisting other colleague Students in the Classroom. He will inform the system about this intent.
- The Student is seeking help from other colleague Students. In this case, he will communicate his intention to the system. The system, in turn, will check if there is a matching Student who can help him.
- The Student is not interested in communicating with other Students. This decision would imply that the Student wants to learn independently, and he is interested in learning only from the Teacher. We shall see that, in some cases, this is advantageous.

5.3.1.1 Formalization of the *Tactic-LA*

In our present instantiation, the *Tactic-LA*, which is used to decide on the action for each Student Simulator in the Student-Classroom interaction, is a 4-tuple: $\{\underline{\alpha}, \underline{\beta}, P, T\}$, where:

$\underline{\alpha} = \{\alpha_1^H, \alpha_2^H, \alpha_3^H\}$, in which

α_1^H is the action corresponding to the decision that he is willing to help others.

α_2^H is the action corresponding to the decision that he is requesting help from others.

α_3^H is the action corresponding to the decision that he wants no interaction.

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

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

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

$P = [p_1^H, p_2^H, p_3^H]^T$, where

$p_i^H(n)$ is the current probability of the interaction decision being represented by α_i^H.

T is the probability updating rule given as a map as:

T: $(P, \underline{\alpha}, \underline{\beta}) \rightarrow P$.

Each of these will now be clarified.

1. $\underline{\beta}$ is the input which the *Tactic-LA* receives from its "Environment", which is now a segment of the overall Tutorial-*like* system. This input is dependent on the reward probabilities that the Environment assigns to each action.

   The selection of the reward probabilities of the *Tactic-LA*'s action, implicitly, defines how we need the Student Simulator to behave while interacting with other Students. Increasing the reward probability of one action will indicate that this action will, in the long term, be used more often as an interaction decision for the Student Simulator. For example, if the overall goal of the Tutorial-*like* system is to enable each Student to learn independently, the reward probability of α_3^H will be assigned as the highest.

   Thus, the action which possesses the highest reward probability should be the one to which the *Tactic-LA* ultimately converges. The system is also permitted the right to tune these reward probabilities. Thus, while it could prefer a certain Student to work independently (for example, if he is an extremely sharp Student), it could set the reward probabilities to force the weaker Students to work in a cooperative manner.

2. P is the action probability vector which contains the probabilities that the *Tactic-LA* assigns to each of *its* actions. At each instant t, the *Tactic-LA*

randomly selects an action $\alpha(t) = \alpha_i^H$. The probability that the automaton selects action α_i^H, at time t, is the action probability $p_i(t) = \Pr[\alpha(t) = \alpha_i^H]$, where $\sum_{i=1}^{r} p_i(t) = 1 \quad \forall\, t$, i=1, 2, 3.

Initially, the *Tactic-LA* will have an equal probability for each of the three interaction decisions as:

$$[0.333 \quad 0.333 \quad 0.333]^{\mathrm{T}}.$$

When the *Tactic-LA* converges to one of its actions, this will indicate that the decisions of the Student in question, in a Classroom-interaction, will be directed to this action.

The special feature about the *Tactic-LA* is that, we are not essentially interested in the convergence of this automaton. We are, mainly, interested in its behavior while it is converging. This behavior mimics the Student interaction within the Classroom *while* he is still trying to learn.

3. T is the updating algorithm, or the reinforcement scheme used to update the *Tactic-LA*'s action probability vector, and is represented by:

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

   This updating algorithm could follow *any* LA learning schemes. For example, if it obeys the DL_{RI} rule, the action probability vector is updated in a *discretized* manner when the *Tactic-LA* is rewarded, and unchanged when the automaton is penalized. Typically, the updating algorithm for the *Tactic-LA* may be selected to have a slow convergence. An ultimate performance criterion for this automaton is that it would converge simultaneously even as the Student is accomplishing his learning of the teaching material.

5.3.2 Formalization of the Student-Classroom Interaction

The teaching material in the Tutorial-*like* system is comprised of multiple choice questions. The Student needs to learn the contents of these questions and conclude

that the answer of each question is the choice that has the highest reward probability. Each Student's LA is thus a 5-tuple:

$\{\underline{\alpha}, \underline{\beta}, \mathrm{P}, T, V\}$, 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.

$\mathrm{P} = [\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_R]^{\mathrm{T}}$, where

$\mathrm{p}_i(\mathrm{n})$ is the current probability that the Student Simulator selects the choice that is represented by α_i.

T is the probability updating rule given as a map as:

T: $(\mathrm{P}, \underline{\alpha}, \underline{\beta}) \rightarrow \mathrm{P}$.

T can be selected to be any LA scheme which defines the learning capacity of the Student. For example, Students, as defined in Chapter 4, can be classified into three types:

- Fast Learners: In this type of Students, T can be defined as a Pursuit scheme.
- Normal Learner: T can use the VSSA scheme to define Students of this type.
- Slow Learner: In this type, T can be defined to use an FSSA scheme or a slower VSSA scheme.

V is the probability updating rule, representing the resultant effect of a Student

communicating with a colleague Student, and this is given as:

V: P×P → P, where,

the first parameter of the map is the action probability vector of the Student concerned, and the second parameter is the action probability vector of the Student providing assistance. These two are combined to yield the updated value of the Student's action probability vector.

When the Student Simulator uses the help from a colleague Student, he uses the latter's action probability vector to improve his knowledge. He can use this knowledge in one of two approaches:

- He *fully* replaces his knowledge with that of the Student providing assistance. In this case, the previous knowledge of the Student will be wiped out, and thus:

  $V(\mathrm{P}, \mathrm{P}^H)=\mathrm{P}$.

- He *partially* utilizes the knowledge provided by the Student who offers assistance. The Student Simulator will use the knowledge of the Student providing the assistance to improve his own as:

  $V(\mathrm{P}, \mathrm{P}^H)=\mathrm{P}'$.

  A simple strategy to assign a value for P′ is as follows:

  - For the "Best" action α_B (whose current probability is p_B), which the Student who is providing help "considers" to be the correct solution:

    $\mathrm{p}_B = \mathrm{p}_B + (\mathrm{p}_B^H - \mathrm{p}_B) / 2$.

  - For all other actions, $i \neq B$:

    $\mathrm{p}_i = \mathrm{p}_i - \Delta$, where $\sum \Delta = (\mathrm{p}_B^H - \mathrm{p}_B) / 2$.

Observe that P′ remains to be a valid probability vector.

5.4 Experimental Results

In this Section, we present the experimental results obtained by testing the prototype implementation of the Student-Classroom interaction. These results were obtained by performing numerous experiments to accurately simulate how a Student interacts with the Teacher and with other colleague Students.

The teaching problems in these experiments have been represented by two different types of Environments, namely two 4-action Environments, and two 10-action Environments, both of which have been earlier used as benchmarks in the field of LA. These Environments are intended to represent the uncertain course material for the Domain model of the Tutorial-*like* system. In all the experiments, an algorithm was considered to have converged if the probability of choosing an action was greater than or equal to a threshold T $(0 < T \leq 1)$ close to unity. Furthermore, an automaton was considered to converge *correctly* if it converged to the action with the highest probability of being rewarded.

The Classroom in the experiments had 9 Students who were capable of interacting with the Teacher and with each other. This Classroom included the following types of Students:

- Three 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. In this scheme, each LA updated its action probability vector if it obtained a reward. Observe that the estimate vector for the reward probabilities was always updated.

- Three 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.

- Three Below-Normal Students ("Slow Learners"). The Student Simulators also

used VSSA[1] 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.

5.4.1 *Tactic-LA* Implementation

Each Student Simulator used a *Tactic-LA* module to decide its strategy as to how it would interact within the Classroom. In our case, the *Tactic-LA* was implemented as a Discretized LA, namely, the DL_{RI} scheme, with a resolution parameter N=200. It was implemented to model the Student's decision as to how to initiate a Student-Student interaction. As explained earlier, each LA had three actions $\{\alpha_1^H, \alpha_2^H, \alpha_3^H,\}$, where α_1^H represents the Student requiring assistance, α_2^H represents the Student being willing to provide assistance, and α_3^H represents a passive non-interacting Student.

The Environment for the *Tactic-LA* was characterized by *its* reward probabilities, which in our experiments had the values:

$$\{0.2 \quad 0.5 \quad 0.7\}.$$

In this case, the optimal settings for any Student is when he is passive. He is rewarded with a probability 0.2 if he requests assistance, and with a probability 0.5 if he provides assistance. Thus, the reward probability value of 0.7 reflects that it is advantageous for the majority of the Students to work independently.

5.4.2 Implementation of Student-Classroom Interaction

In our experiments, the sharing of information and knowledge in the Student-Classroom interaction was made possible by the Helper Student who provided his action probability vector to the one who requested help. The processing of the information provided was achieved by one of the following two scenarios:

[1]One of the goals of this research was to also model slow learning using FSSA, for example, by means of Tsetlin's or Krinsky's automaton. While this has been achieved for Student *modeling*, as described in Chapter 4, the problem of using them for Classroom interaction is still open.

- Complete transfer of knowledge. In this scenario, the Student who required assistance replaced his action probability vector with the one provided by the Student providing assistance. In this case, all the prior knowledge of the Student was erased, and replaced with the knowledge offered.

- Partial utilization of knowledge. In this scenario, the Student requiring assistance partially used the knowledge of the Helper Student. Thus, he improved his knowledge incrementally, utilizing the action probability vector suggested by his Helper Student. To achieve this, the Student performed the following steps:

  - By examining the action probability vector of the Helper Student, he determined the action that has the highest probability. He then considered this action to be the one that was recommended by the Helper Student.
  - The Student updated his own action probability vector as shown below:
    * For the action that corresponds to the Helper Student's recommended action, he modified the probability value towards that which was suggested by the Helper Student. Observe that this could either increase or decrease the previous probability of this action. In our implementation, we forced the value to go to a value half-way from its current value to the proposed value.
    * For all other actions, the action probabilities were updated so that the action probability vector continued to be a valid probability vector.

The Student Simulator was implemented to use four types of strategies for the Student-Classroom interaction, as described earlier in Section 5.2:

1. In the first strategy, there was no interaction between the Students. Thus, since the Students opted to not communicate with each other (and share information), they were only dependent on the Teacher. The simulation results proposed for this scenario were considered as the base-line results, and were used to compare

how other Student-Classroom interaction strategies improved the learning of the Student.

2. In the second strategy, the *Tactic-LA* of the Student Simulator periodically selected an action. In our implementation, this was done after every 10 steps. If the selected action recommended requesting assistance from another Student, the Student Simulator contacted the Communication Agent and requested assistance. When the Communication Agent provided him with another Student in the Classroom who was willing to help, the Student Simulator considered the information provided by the latter to be reliable. He then used this acquired knowledge to either completely replace or to partially improve his knowledge.

3. The third strategy used in our experiments for the Student-Classroom interaction was similar to the second. However, in this case, before the Student received help from another Student, he evaluated the Helper's knowledge. The Student Simulator examined the action probability vector of the Helper Student, and decided that the Helper Student possessed a superior knowledge if the convergence state of the latter's action probability vector was apparently closer to unity than that of himself. The knowledge provided by the Helper Student was utilized only if this was the case.

4. The fourth strategy can be perceived to be an extension of the second. However, in this case, the Student in question did the additional task of examining the learned knowledge after a probationary period. If the Student Simulator in question discovered that his knowledge did not improve during this probationary period, he "unlearned" what he was taught by his Helper Student by purging his system from the knowledge that he obtained from this colleague.

The simulation experiments were performed for the various existing benchmark Environments with 4 actions and 10 actions, for which the threshold, T, was set to be 0.99, and the number of experiments, NE, was 75. The results of these simulations are described below.

5.4.3 Results using 4-Action and 10-Action Environments

The experiments were done using the two sets of benchmark Environments, namely, the two 4-action Environments ($E_{4,A}$ and $E_{4,B}$), and the two 10-action Environments ($E_{10,A}$ and $E_{10,B}$), introduced earlier in Section 4.6.1 and Section 4.6.2. Again, the 4-action Environment represents a multiple-choice question with 4 options, whereas the 10-action Environment represents a more difficult multiple-choice question with 10 options. The nine Students in the simulations needed to learn the responses for the questions, and were asked to determine the corresponding answers, which in this model represented the actions which possessed the minimum penalty probability. The simulation of the Student-Classroom interaction would reveal if the Students benefited from their interaction with other Students according to their interaction strategy.

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.

In the following sections, we will provide, in detail, the learning characteristics of the Students when they resorted to the different strategies in their Student-Classroom interaction.

Env.	No. of actions	Student Type	No Interaction between Students	
			No. iterations to converge	% of incorrect convergence
$\mathbf{E_{4,A}}$	4	Fast Learner	572	0
		Normal Learner	996	0
		Below-Normal Learner	1382	0
$\mathbf{E_{4,B}}$	4	Fast Learner	1482	0
		Normal Learner	2201	0
		Below-Normal Learner	2633	0
$\mathbf{E_{10,A}}$	10	Fast Learner	686	1
		Normal Learner	1297	1
		Below-Normal Learner	1804	0
$\mathbf{E_{10,B}}$	10	Fast Learner	1655	4
		Normal Learner	2114	4
		Below-Normal Learner	2859	4

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 5.1: Convergence of the Student Simulators learning only from the Teacher, in the Benchmark 4/10-Action Environments.

5.4.3.1 Transfer No Knowledge

In the first interaction strategy, the Students opted not to share information between each other. The number of iterations that was needed for the Students to learn the teaching material were based only on the interaction between the Student and the Teacher. The results of this strategy are presented in Table 5.1.

The experimental results of this strategy represented a base-line, using which the performance of the Student using the other strategies could be compared. The results showed that the convergence for the $E_{4,B}$ and $E_{10,B}$ Environments took longer than

the time required for the $E_{4,A}$ and $E_{10,A}$ Environments respectively. This reflects the fact that the set of E_B Environments were more difficult because of the proximity of the underlying penalty/reward probabilities.

5.4.3.2 Always Consider the Other Student's Knowledge as Reliable

In the second strategy, when a Student Simulator requested to get assistance from other Students, he assumed that the Helper Student's knowledge was credible. Thus, he made use of the Helper Student's knowledge in one of two possible ways, as detailed below.

Completely Replace Knowledge: In the first approach, the Student Simulator completely replaced his knowledge with the knowledge of the Helper Student. The simulation results of this interaction are tabulated in Table 5.2. The results show that the learning of the Fast Students deteriorated, while the learning for the Normal and Below-Normal Students improved. For example, in the 4-action $E_{4,A}$ Environment, learning for the Fast Student declined by 30% as the iterations needed for his LA to converge increased from 572 to 741. The number of iterations needed for Normal Student LA to converge decreased from 996 to 730, which was a 27% increase in the learning performance. As for the Below-Normal Student, the iterations required for his LA to converge decreased from 1,382 to 768, which was a 44% improvement in the learning. Of course, the deterioration of the learning of the Fast Student can be attributed to the fact that in the Classroom, there were 3 Fast Students and 6 Normal and Below-Normal Students. Thus, when a Fast Student was looking for help, he could get help from either 2 of his colleagues *en par* with him, or 6 who were technically inferior. Clearly, the possibility that he would get help from a weaker Student was larger.

Similarly, in the 10-action $E_{10,A}$ Environment, the learning of the Fast Student deteriorated by 37%, while the learning for the Normal and Below-Normal Students improved by 28% and 46% respectively.

Env.	Student Type	Always consider the help provided					
		No. iter. to conv.	% gain /decline	% wrong conv.	No. fast interac.	No. normal interac.	No. below normal interac.
$E_{4,A}$	Fast Learner	741	↓30	0	4	8	6
	Normal Learner	730	↑27	0	8	3	5
	Below-Normal Learner	768	↑44	0	7	7	3
$E_{4,B}$	Fast Learner	1691	↓14	0	8	9	10
	Normal Learner	1617	↑27	0	10	6	8
	Below-Normal Learner	1717	↑35	0	10	9	7
$E_{10,A}$	Fast Learner	943	↓37	0	4	9	7
	Normal Learner	932	↑28	0	10	3	6
	Below-Normal Learner	971	↑46	0	9	7	3
$E_{10,B}$	Fast Learner	1765	↓7	0	8	9	9
	Normal Learner	1762	↑17	0	10	7	9
	Below-Normal Learner	1899	↑34	0	10	10	6

Table 5.2: Convergence of the Student Simulators in a Student-Classroom interaction which *always* recommended the use of the knowledge from the Helper Students.

From an overall perspective, ironically, we can conclude that with an interaction of *this* type, Normal Students were the ones who tended to learn even faster than the "Fast" learners, because they had a higher chance of getting assistance from Fast Students or their equals. However, Below-Normal Students were the ones who *benefited* the most from this strategy. This was because they almost always got assistance from superior colleagues.

Partially Utilize Knowledge: In the second approach, the Student Simulator opted to partially utilize the knowledge from the Helper Student. The simulation results of this interaction are given in Table 5.3. The results were similar to those obtained in the first approach (i.e., one which recommended the complete transfer of knowledge), in that the learning of Fast Students declined, while the learning for Normal and Below-Normal Students improved. For example, in the 4-action $E_{4,A}$ Environment, the learning for the Fast Students declined by 58% since the number of iterations required for the LA to converge increased from 572 to 904. The number of iterations required for Normal Students' LA convergence decreased marginally from

996 to 970, which represented only a 3% increase in their learning performance. As for the Below-Normal Students, the number of iterations for their LA convergence decreased from 1,382 to 1,118, which is a 19% improvement in their learning. Again, the deterioration of the learning of Fast Students can be attributed to the fact that they usually got information from weaker colleagues.

Similar results were recorded for the 10-action Environments. For example, for the 10-action $E_{10,A}$, learning for the Fast Student deteriorated by 68%, while the learning for the Normal Students and Below-Normal Students improved by 6% and 20% respectively.

With this approach, Fast Students remained to be the fastest in learning, which was different from the observations made in the first approach, namely the one in which the Students completely replaced their knowledge with the Helper Students' knowledge. We believe that the reason for this is the following: When a Student completely replaced his knowledge with that of the Helper Student, he was totally dependent on the other Student's knowledge. Thus, the learning ability of Fast Students would be influenced more with knowledge provided by inferior Students. However, when the knowledge transfer was partial, the Student also used the information he gleaned from his previous experience and learning. This is, of course, most beneficial to Fast Students.

We should highlight, though, that neither of these approaches had any impact on the accuracy of the learning. All types of Students still reached the correct conclusion 100% of the time.

5.4.3.3 Incorporate the Other Student's Knowledge *only* if it is Reliable

In the third strategy of the Student-Classroom interaction, the Student, when requesting help and if he could find a colleague who was willing to help him, evaluated the Helper Student's knowledge. If he considered the latter's knowledge to be better than his, he would use that knowledge to improve his own. Again, there were

Env.	Student Type	Always consider the help provided (partial)					
		No. iter. to conv.	% gain /decline	% wrong conv.	No. Fast interac.	No. Normal interac.	No. Below Normal interac.
$E_{4,A}$	Fast Learner	904	↓58	0	5	8	8
	Normal Learner	970	↑3	0	7	5	7
	Below-Normal Learner	1118	↑19	0	8	8	7
$E_{4,B}$	Fast Learner	1968	↓33	0	9	11	11
	Normal Learner	2112	↑4	0	12	8	10
	Below-Normal Learner	2429	↑8	0	10	10	7
$E_{10,A}$	Fast Learner	1150	↓68	0	7	8	9
	Normal Learner	1225	↑6	0	10	5	9
	Below-Normal Learner	1443	↑20	0	9	9	7
$E_{10,B}$	Fast Learner	2071	↓25	0	8	11	11
	Normal Learner	2075	↑2	0	11	7	10
	Below-Normal Learner	2393	↑16	0	11	11	7

Table 5.3: Convergence of the Student Simulators in a Student-Classroom interaction which *always* recommended the *partial* use of the knowledge from the Helper Students.

two approaches which the Student could resort to make use of the Helper Student's knowledge.

Completely Replace Knowledge: In the first approach, the Student completely replaced his knowledge with that of the Helper Student. The results of this interaction are given in Table 5.4. The results showed that *all* types of Students benefited from this strategy. For example, in the 4-action $E_{4,A}$ Environment, a Fast Student improved his learning by absorbing the teaching material in 492 iterations instead of 572 iterations, which is a 14% improvement. For a Normal Student, his learning improved by reaching the conclusion in 476 iterations instead of 996 iterations, which is a 52% improvement. Finally, for a Below-Normal Student, his knowledge improved by learning in 507 iterations instead of 1,382 iterations, which is a 63% improvement.

In this approach, ironically, again Normal Students tended to learn faster than even the usually "Fast" Students. This again can be attributed to the fact that a Normal Student can interact with three Fast Students, while a Fast Student can

interact only with two Fast colleagues. Below-Normal Students were the ones who benefited the most in this approach, with a learning improvement of a 63%, as they interacted more with superior colleagues.

However, the results showed that this approach sometimes caused a marginal number of Students to reach incorrect results in some cases, especially for the more difficult Environments $E_{4,B}$ and $E_{10,B}$. For example, in the 4-action $E_{4,B}$ Environment, Normal and Below-Normal Students suffered from learning incorrect results in 9% of the cases. Similarly, in the 10-action $E_{10,B}$ Environment, a Fast Student yielded incorrect learning results 1% of the time, while Normal and Below-Normal Students suffered from inaccurate learning 13% of the time. While it is hard to pinpoint the cause for this, it could be attributed to the fact that if a Normal, or Below-Normal Student, incorrectly started to converge to an incorrect result, rectifying this erroneous "trajectory" with such a strategy was difficult. The latter is true because such a strategy would encourage, in some cases, the Student to continue the "deviant" path. Thus, since Fast Students usually converged to the correct answer from the early iterations, we see that it is more difficult to lead such Students to a wrong answer. As a further observation, the fact that the number of incorrect results was more in the E_B Environments could be attributed to the premise that these Environments were more difficult than those of the E_A type.

Partially Utilize Knowledge: In the second approach, the Student utilized the knowledge of the Helper Student only in a *partial* manner, to incrementally improve his knowledge. The results of this approach are tabulated in Table 5.5. These results showed a slight (or marginal) improvement in the learning of the Fast Students, and a *significant* improvement in the learning of the Normal and Below-Normal Students. For example, for the 4-action $E_{4,A}$ Environment, a Fast Student improved his learning by converging to the correct answer in 551 iterations, as opposed to 572 iterations (the base-line case), which is a marginal 4% improvement. On the other hand, a Normal Student improved his learning by converging to the correct answer in 777 iterations instead of 996 iterations, which is a 22% improvement, and for a Below-Normal Student, the improvement was even higher as he reached the correct answer

Env.	Student Type	Consider help if better than current knowledge					
		No. iter. to conv.	% gain /decline	% wrong conv.	No. Fast interac.	No. Normal interac.	No. Below Normal interac.
$E_{4,A}$	Fast Learner	492	↑14	0	2	2	1
	Normal Learner	476	↑52	0	3	2	2
	Below-Normal Learner	507	↑63	0	4	3	2
$E_{4,B}$	Fast Learner	972	↑34	0	2	4	4
	Normal Learner	879	↑60	9	2	2	2
	Below-Normal Learner	758	↑71	9	3	3	2
$E_{10,A}$	Fast Learner	585	↑15	0	2	2	1
	Normal Learner	570	↑56	1	3	2	2
	Below-Normal Learner	586	↑68	1	4	4	2
$E_{10,B}$	Fast Learner	1053	↑36	1	3	4	4
	Normal Learner	780	↑63	13	3	2	3
	Below-Normal Learner	784	↑73	13	3	4	2

Table 5.4: Convergence of the Student Simulators in a Student-Classroom interaction which recommended the use of the knowledge from the Helper Students if it was deemed "reliable".

in 856 iterations instead of 1,382 iterations, which is a 38% gain. Similar results were also recorded in the 10-action $E_{10,A}$ Environments, where in the best case, the learning improved by 39%.

Although these results led to a slightly slower convergence than in the first approach (which had a complete replacement of information), the experiments did not erase the previous experience of the Students, but attempted to build on it. This also resulted in a more accurate learning. This can be observed from the results obtained in all Environments, where *all* Students learned the correct knowledge 100% of the time !!

5.4.3.4 "Unlearn" if the Student's Knowledge does not Improve

In the fourth and last strategy, the Student, after a probationary period, "unlearned" the knowledge from the Classroom interaction if his knowledge did not improve.

Env.	Student Type	Consider help if better than current knowledge (partial)					
		No. iter. to conv.	% gain /decline	% wrong conv.	No. Fast interac.	No. Normal interac.	No. Below Normal interac.
$E_{4,A}$	Fast Learner	551	↑4	0	2	1	0
	Normal Learner	777	↑22	0	5	2	3
	Below-Normal Learner	856	↑38	0	5	4	3
$E_{4,B}$	Fast Learner	1290	↑13	0	3	4	3
	Normal Learner	1511	↑31	0	7	2	3
	Below-Normal Learner	1593	↑39	0	8	5	2
$E_{10,A}$	Fast Learner	656	↑4	0	2	1	0
	Normal Learner	959	↑26	0	5	3	4
	Below-Normal Learner	1095	↑39	0	5	6	4
$E_{10,B}$	Fast Learner	1450	↑12	0	3	3	2
	Normal Learner	1704	↑19	0	7	3	4
	Below-Normal Learner	1961	↑31	0	6	6	3

Table 5.5: Convergence of the Student Simulators in a Student-Classroom interaction which recommended the *partial* use of the knowledge from the Helper Students if it was deemed "reliable".

Again, the simulation experiments of this strategy were done by utilizing the Helper Students' knowledge in two approaches, i.e., by either completely replacing or partially utilizing the knowledge provided.

Completely Replace Knowledge: In the first approach, the Student completely replaced his knowledge with that of his colleague Student, prior to deciding whether he would utilize this knowledge or not. The results of this approach are given in Table 5.6.

In this approach, the learning for Fast Students deteriorated, while the learning for the Normal and Below-Normal Students improved. For example, in the 4-action $E_{4,A}$ Environment, the learning of the Fast Students declined, as seen in the number on iterations increasing from 572 to 789 – a decline of 38%. For a Normal Student, his learning improved by 26% – the number of iterations needed decreased from 996 to 742 iterations. Finally, the gain of the Below-Normal Students was the highest,

Env.	Student Type	Unlearn if knowledge does not improve						
		# iter. to conv.	% gain /decline	% wrong conv.	# Fast interac.	# Norm. interac.	# Below Normal interac.	# of unlearn
$E_{4,A}$	Fast Learner	789	↓38	0	5	6	7	1
	Normal Learner	742	↑26	0	7	4	7	3
	Below-Norm. Learn.	809	↑41	0	7	6	5	3
$E_{4,B}$	Fast Learner	1603	↓8	0	7	9	9	2
	Normal Learner	1506	↑32	0	9	6	8	6
	Below-Norm. Learn.	1576	↑40	0	10	9	7	7
$E_{10,A}$	Fast Learner	935	↓36	0	5	7	8	1
	Normal Learner	918	↑29	0	8	5	7	4
	Below-Norm. Learn.	958	↑47	0	8	8	5	4
$E_{10,B}$	Fast Learner	1724	↓4	0	7	9	9	4
	Normal Learner	1676	↑21	0	10	7	9	7
	Below-Norm. Learn.	1785	↑38	0	10	10	7	7

Table 5.6: Convergence of the Student Simulators in a Student-Classroom interaction which initially recommended the complete use of the knowledge from the Helper Students, but where this knowledge can be *unlearned* if warranted.

as their learning improved by 41%, decreasing the number of iterations needed from 1,382 to 809. Again, the rational for deterioration of Fast Students is probably the same as in the previous cases.

Similar results were also recorded for the 10-action Environments. For example, in the $E_{10,A}$ Environment, the learning of Fast Students deteriorated by 36%, while the learning for both Normal and Below-Normal Students improved by 29% and 47% respectively.

Partially Utilize Knowledge: In the second approach for this strategy, the Student partially used the knowledge from the Helper Student to improve his own knowledge before deciding, after the probationary period, whether to use the information or not. The results of this approach are given in Table 5.7.

In this approach, as in previous cases, the Fast Students' learning deteriorated,

Env.	Student Type	Unlearn if knowledge does not improve (partial)						
		# iter. to conv.	% gain /decline	% wrong conv.	# Fast interac.	# Norm. interac.	# Below Normal interac.	# of unlearn
$E_{4,A}$	Fast Learner	854	↓49	0	5	7	7	0
	Normal Learner	851	↑15	0	7	4	7	3
	Below-Norm. Learn.	972	↑30	0	7	8	5	4
$E_{4,B}$	Fast Learner	1720	↓16	0	8	10	9	1
	Normal Learner	1645	↑25	0	11	6	10	7
	Below-Norm. Learn.	1791	↑32	0	10	9	7	7
$E_{10,A}$	Fast Learner	1062	↓55	0	6	8	9	1
	Normal Learner	1102	↑15	0	9	6	9	4
	Below-Norm. Learn.	1242	↑31	0	8	9	7	4
$E_{10,B}$	Fast Learner	1943	↓17	0	8	10	10	2
	Normal Learner	1961	↑7	0	11	8	10	7
	Below-Norm. Learn.	2040	↑29	0	11	10	7	7

Table 5.7: Convergence of the Student Simulators in a Student-Classroom interaction which initially recommended the *partial* use of the knowledge from the Helper Students, but where this knowledge can be *unlearned* if warranted.

while the Normal and Below-Normal Students improved. For example, in the 4-action $E_{4,A}$ Environment, the Fast Students learning declined by 49% (needing 854 iterations instead of 572). Normal Students improved their learning by 15%, requiring only 851 iterations instead of 996, and finally, the Below-Normal Students improved their learning by the highest percentage, 30%, requiring 972 iterations instead of 1,382.

The results for the 10-action Environments are analogous. For example, in the $E_{10,A}$ Environment, the learning for Fast Students deteriorated by 55%, while the learning for both Normal and Below-Normal Students improved by 15% and 31% respectively.

For all Student-Classroom interaction strategies, the simulation experiments showed that most Students benefited from the interaction. Below-Normal and Normal students *always* improved their learning while interacting with other fellow Students. It

Type	**Below-Normal** Learner	**Normal Learner**	**Fast Learner**
Always completely replace	3	4	–
Always partially utilize	6	6	–
Completely replace if reliable	1	1	1
Partially utilize if reliable	4	3	2
Completely replace, then may unlearn	2	2	–
Partially utilize, then may unlearn	5	5	–

Table 5.8: Overall ranking of the different strategies in a Student-Classroom setting which resulted in improving the learning characteristics of Students. In the case of the Fast learner, all the other interaction strategies, other than those ranked, proved to yield a negative impact on his learning.

was only the set of Fast Students, who, in some strategies, did worse than when they worked independently.

Based on our experiments, the following table, viewed from an overall perspective, encapsulates our evaluation of how a Student-Classroom interaction can best be optimized. We conclude this section by briefly remarking that this ranking is almost identical to what we, as intelligent humans, would do !

5.5 Conclusion

This Chapter presented a novel paradigm for a Tutorial-*like* system in which a Student is a member of a Classroom of Students. The Students are permitted to not only learn from the Teacher but also from any of their colleague Students in the Classroom. This paradigm is a departure from the general learning LA paradigm, where a Student learns only from a Teacher, or from multiple Teachers.

The Chapter presented a new class of interconnecting LA, in which different Students (LA) are interacting, and where each Student is trying to maximize his learning. The Students were, collectively, trying to improve their learning by interacting

with each other. To enable this interaction, Student Simulators utilize a so called *Tactic-LA* which enables the Students to determine the actions needed to facilitate the interaction between themselves.

The Student Simulators used four different strategies in their interactions with each other. These strategies defined the level and the capacity of the interaction between themselves. The main result that we have shown is that a weaker learner can benefit from this interaction by utilizing the knowledge he gets from a superior learner. Although this is obvious, we have shown, unequivocally, how this can be achieved in a LA setting, and how poor information can be discarded.

Further, from the simulation results, we also conclude that the interaction between the different Students was most beneficial to weaker learners – sometimes by about 73%. Superior Students showed either minimal gains or a deterioration in learning, primarily because they more often interacted with weaker Students. We categorically state that within our learning paradigm, our proposed interaction model established that a Student-Classroom interaction is beneficial to the learning process.

In the next Chapter, we will present how to model Socratic type Domain knowledge. The Domain model will represent a teaching Environment with information stored in different *chapters*. The complexity of the knowledge increases in subsequent chapters.

Chapter 6

Modeling the Domain Knowledge

The aim of this Chapter 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. We also address the issue of keeping the identity of the superior actions hidden as the complexity of the problems increase.

To the best of our knowledge, the concept of modeling teaching material with increasing difficulty using an LA paradigm is unique. The main results we have obtained demonstrate that: (a) Increasing the difficulty of the teaching material can

affect the learning of Normal and Below-Normal Students, by resulting in an increased learning time, and (b) This increase 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%.

6.1 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 [8]. 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 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 Chapter 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 later in the Chapter, 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.

6.1.1 Contributions of this Chapter

This Chapter 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 Chapter 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.

6.2 The Domain Model in the Overall Context

6.2.1 The Student/Teacher/Domain in a Tutorial-*like* System

All along, we have argued that in Tutorial-*like* systems, Students (or Student Simulators) try to learn some Domain knowledge from the Teacher and from the interaction

between themselves. As we know, there are no *real-life* Students who use the Tutorial-*like* system. Students are modeled using Student Simulators, that try to mimic the actions and behavior of *real-life* Students. These Student Simulators are, in turn, modeled using LA, which attempt to learn the Domain knowledge from the Teacher, who may also be a modeled entity.

We explained in Chapter 4, that the Tutorial-*like* system would itself model the Students by observing their behavior while using the system and examining how they learn. There we showed that the Student modeler is capable of inferring 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, which is more relevant to what this present topic concerns.

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. Since we work within the LA paradigm, the questions which we encounter, 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?

The crucial issue that this section considers is that of modeling the Domain itself to consider "chapters" of increasing complexity. This will be addressed and formalized in the next section. Thereafter, we can also consider the problem of the Teacher improving his teaching capability – since he has access to a formal "Domain model".

6.2.2 Domains/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[1] 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

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

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.

When 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[2].

6.3 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 simulation experiments were performed using the two sets of benchmark Environments, namely, the two 4-action Environments ($E_{4,A}$ and $E_{4,B}$), and the two 10-action Environments ($E_{10,A}$ and $E_{10,B}$), introduced earlier in Section 4.6.1 and Section 4.6.2. Also, the three types of Students (i.e., the Fast, Normal, and Below-Normal), that were used, were similar to the ones used in Section 5.4. Again, 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.

[2]The question of how to deal with chapters of increasing complexity is dealt with in a subsequent chapter.

Env.	No. of actions	Chapter	μ (diffic. index)	No. iterations for **Fast Stud.** to converge	No. iterations for **Normal Stud.** to converge	No. iterations for **Below-Norm. Stud.** to converge
$\mathbf{E_{4,A}}$	4	1	1.0	563	975	1,380
		2	0.9	542	1,051	1,497
		3	0.8	530	1,192	1,628
		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
$\mathbf{E_{4,B}}$	4	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
		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
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		

Table 6.1: Convergence of the Student Simulators learning in the Benchmark 4-Action Environments by increasing the level of difficulty in the Domain knowledge.

6.3.1 Results using 4-Action and 10-Action Environments

The Students in the simulations, who communicated with the Teacher and learned the teaching material, needed to learn the responses for the questions and determine the choice that possessed the minimum penalty probability. 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 the simulations for the 4-action Environments are provided in Table 6.1, while the results for the 10-action Environments are tabulated in Table 6.2.

The results for the 4-action Environments demonstrate that the difficulty of the

Env.	No. of actions	Chapter	μ (diffic. index)	No. iterations for **Fast Stud.** to converge	No. iterations for **Normal Stud.** to converge	No. iterations for **Below-Norm. Stud.** to converge
$\mathbf{E_{10,A}}$	10	1	1.0	680	1,320	1,728
		2	0.9	684	1,442	1,972
		3	0.8	660	1,485	2,256
		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
$\mathbf{E_{10,B}}$	10	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
		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 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 6.2: Convergence of the Student Simulators learning in the Benchmark 10-Action Environments by increasing the level of difficulty in the Domain knowledge.

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 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 iterations 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 was almost constant. This seems to also be the case for *real-life* Students.

For the 10-action Environments, the results were similar to that of the 4-action Environments. For example, in the $E_{10,A}$ Environment, a Normal learner Student LA learned the material by converging in 1,320 iterations in an Environment without any enhanced level of difficulty (μ=1.0). When μ decreased to 0.7, the Normal Student LA converged in 1,731 iterations, which represents an increase of 31% in the learning time for the Student. When μ was set to be 0.4 to increase the difficulty, the iterations needed for learning increased to 2,604, which represents an increase of 97% of the time required to learn when compared to the original benchmark Environment.

Again, similar results were also recorded for the Below-Normal learner. He learned

the material in 1,728 iterations in an Environment without any enhanced difficulty, while this number increased to 2,386 when the value of μ was set to 0.7, which represents an increase of 38% in the learning time. When the control parameter μ decreased to 0.4, the learning time increased to 3,965 iterations, which represents a 129% increase in the learning time from the original benchmark.

Analogous results were also seen for Fast learners, where they were not adversely affected by increasing the difficulty in the Environment. It was observed that the number of iterations needed to learn the teaching material was almost constant.

Figure 6.1 and 6.2 depicts graphically the results of the simulations for the 4-action and the 10-action benchmark Environments respectively. 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.

6.4 Conclusion

This Chapter 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 Chapter 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

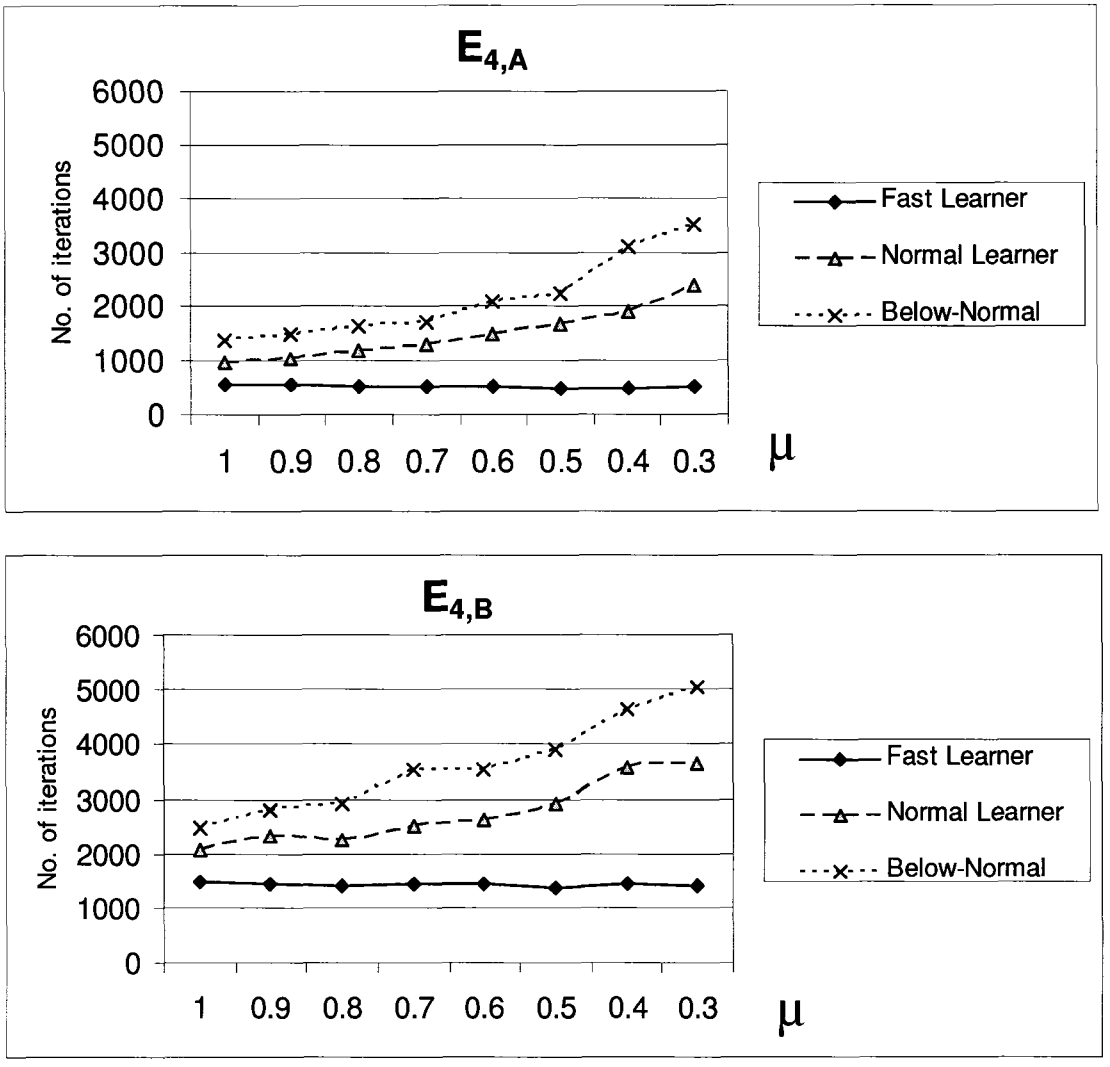

Figure 6.1: The effect of increasing the difficulty of the Domain model on the different types of Students in Benchmark 4-Action Environments.

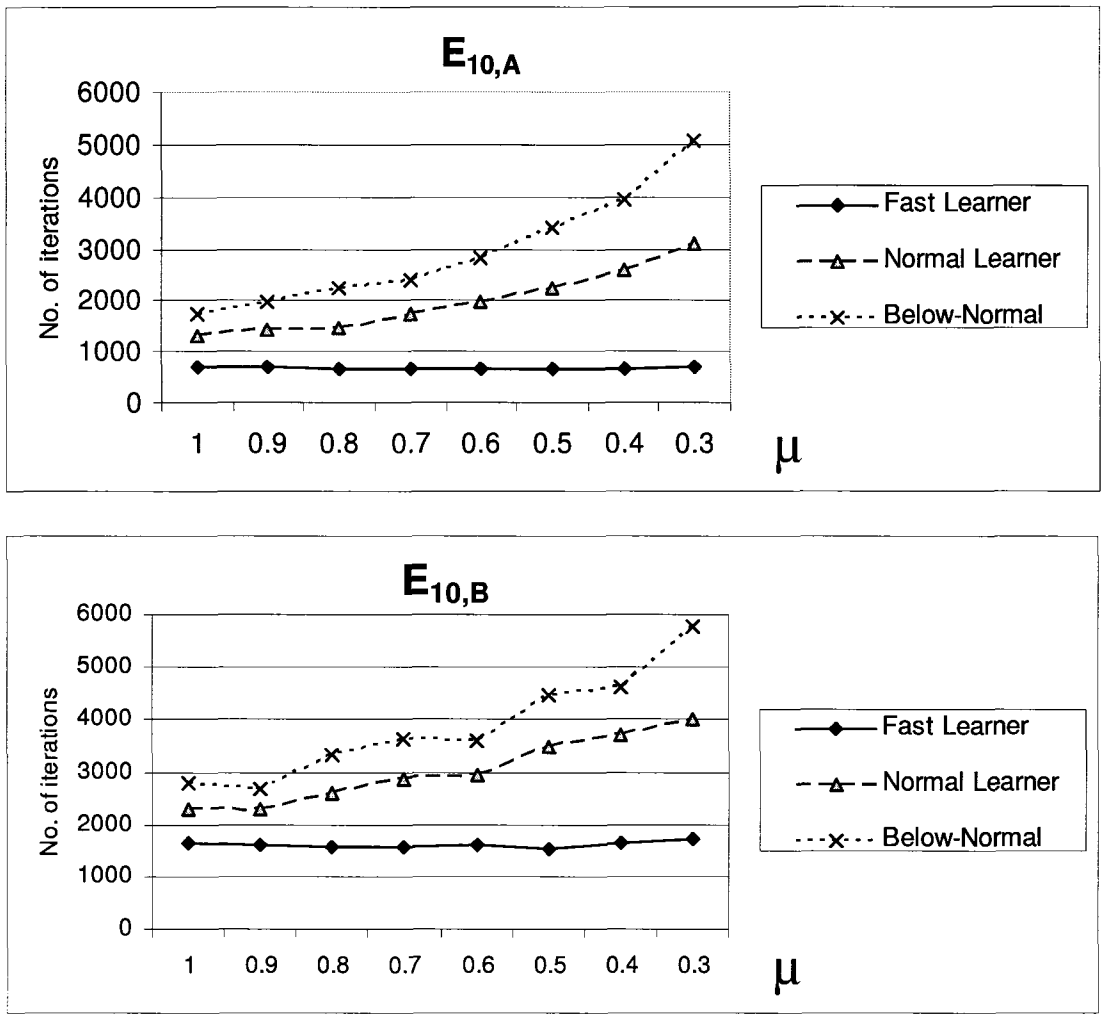

Figure 6.2: The effect of increasing the difficulty of the Domain model on the different types of Students in the Benchmark 10-Action Environments.

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 will be discussed in a subsequent chapter. In the next Chapter, we will present how the Teacher can assist the Students so that they can be able to handle the learning of more complex material. There, we will present the concept of the Teacher providing *hints* to the Students, which will enable them to cope with learning more complex Domain knowledge.

Chapter 7

Modeling the Teacher's Behavior

The goal of this Chapter 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 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 in the field of 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. In our approach, the convergence of the LA (Students) 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.

7.1 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 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 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 Chapter is to 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 Socratic-type knowledge, 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 has 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 became 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 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.

7.1.1 Contributions of this Chapter

This Chapter 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 Chapter 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.
- The concept of a Teacher providing *hints* to the Students so as to assist them in handling more complex information.

7.2 Concept of Teachers who Provide *Hints*

7.2.1 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:

$$[\frac{1}{R} \quad \frac{1}{R} \quad \cdots]^{\mathrm{T}}.$$

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:

  $$p_j = p_j + \rho.(\frac{R-1}{R}).$$

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

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

- 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.

7.2.2 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 believes 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.

7.3 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 was to be taught to the Students. The simulations were performed against the different benchmark Environments, namely, the two 4-action Environments, and the two 10-action Environments, used earlier in Section 4.6.1 and

Section 4.6.2. Again, the simulations were performed (for the different benchmark Environments), with the threshold, T, set to be 0.99, and the number of experiments, NE, = 75. The results of these simulations are described below.

As mentioned earlier, three types of Students have been used in the simulations, who communicated with the Teacher to learn the subject matter. These three types of Students (i.e., the Fast, Normal, and Below-Normal), that were used, were similar to the ones used in Section 5.4.

7.3.1 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 7.2.1.

- 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).

7.3.2 Results using 4-Action and 10-Action Environments

The simulation experiments were performed using the two sets of benchmark Environments, namely, the two 4-action Environments ($E_{4,A}$ and $E_{4,B}$), and the two 10-action Environments ($E_{10,A}$ and $E_{10,B}$). 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.

The results of these simulations are tabulated in Tables 7.1-7.4 for Environments $E_{4,A}$, $E_{4,B}$, $E_{10,A}$, and $E_{10,B}$ respectively.

7.3.2.1 $E_{4,A}$ Environment

The results for the $E_{4,A}$ Environment are given in Table 7.1. 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 $\rho > 0.3$. 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 7.1. 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.

7.3.2.2 $E_{4,B}$ 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 7.2.

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 iterations 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.

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					

Table 7.1: 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 7.2.2).

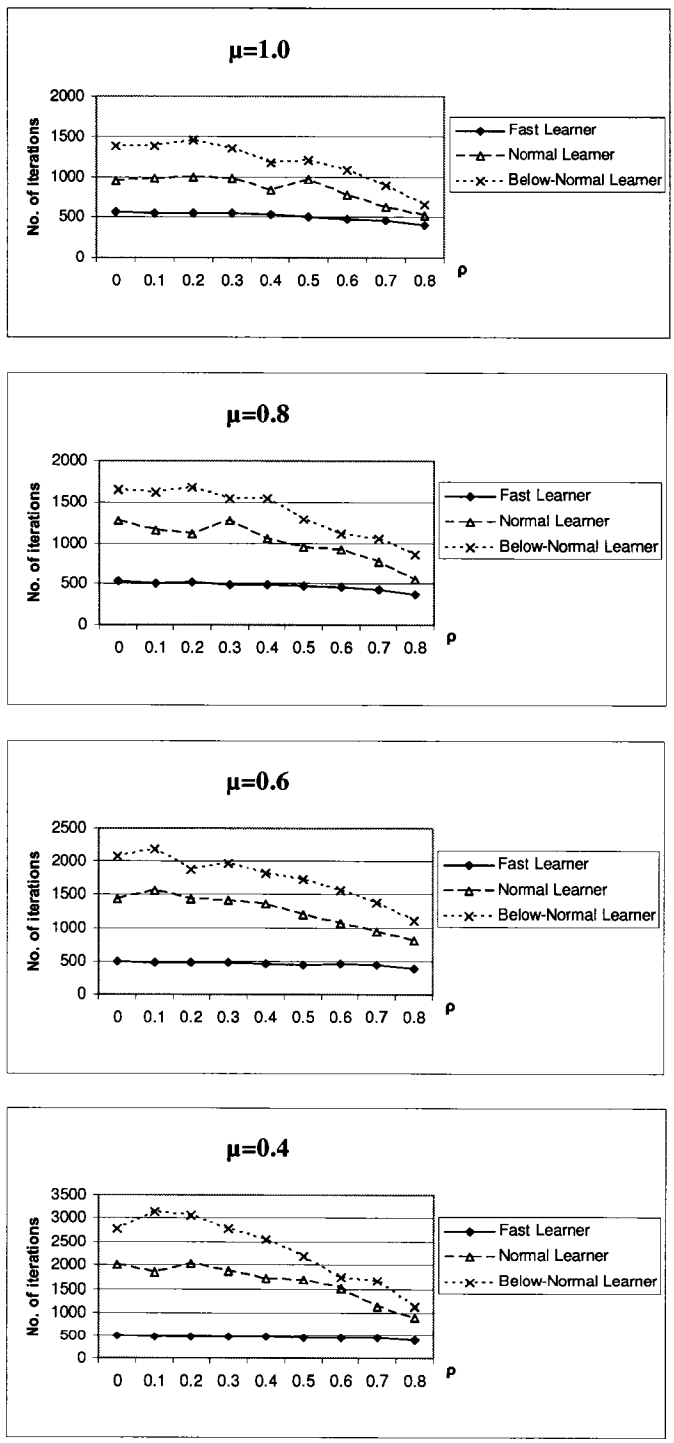

Figure 7.1: The effect of increasing the *Hint Index* provided by the Teacher on Students learning in Environments derived from $E_{4,A}$ by varying the *Difficulty Indices*.

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 7.2. 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.

7.3.2.3 $E_{10,A}$ Environment

In the interest 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 7.3.

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 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)

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

Table 7.2: 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 7.2.2).

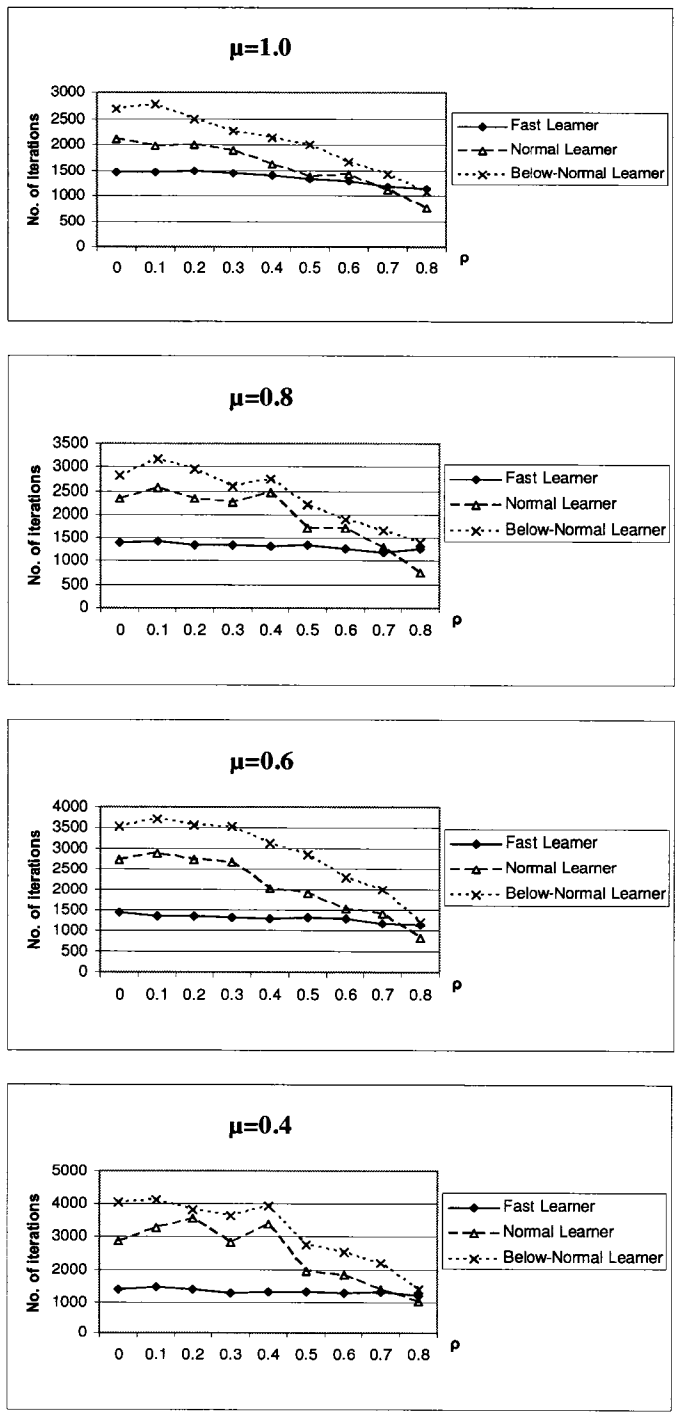

Figure 7.2: The effect of increasing the *Hint Index* provided by the Teacher on Students learning in Environments derived from $E_{4,B}$ by varying the *Difficulty Indices*.

to 591 iterations.

The results of the $E_{10,A}$ Environment simulations are depicted graphically in Figure 7.3. 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 $E_{10,B}$ Environment. The simulation results of this Environment are tabulated in Table 7.4, and depicted graphically in Figure 7.4. 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.

7.4 Conclusion

This Chapter 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.

In the next Chapter, we will present how the Teacher *himself* will be able to

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	0.10	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
1.0	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
	0.0	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
0.8	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
	0.0	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
0.6	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
	0.0	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	0.4	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		

Table 7.3: 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 7.2.2).

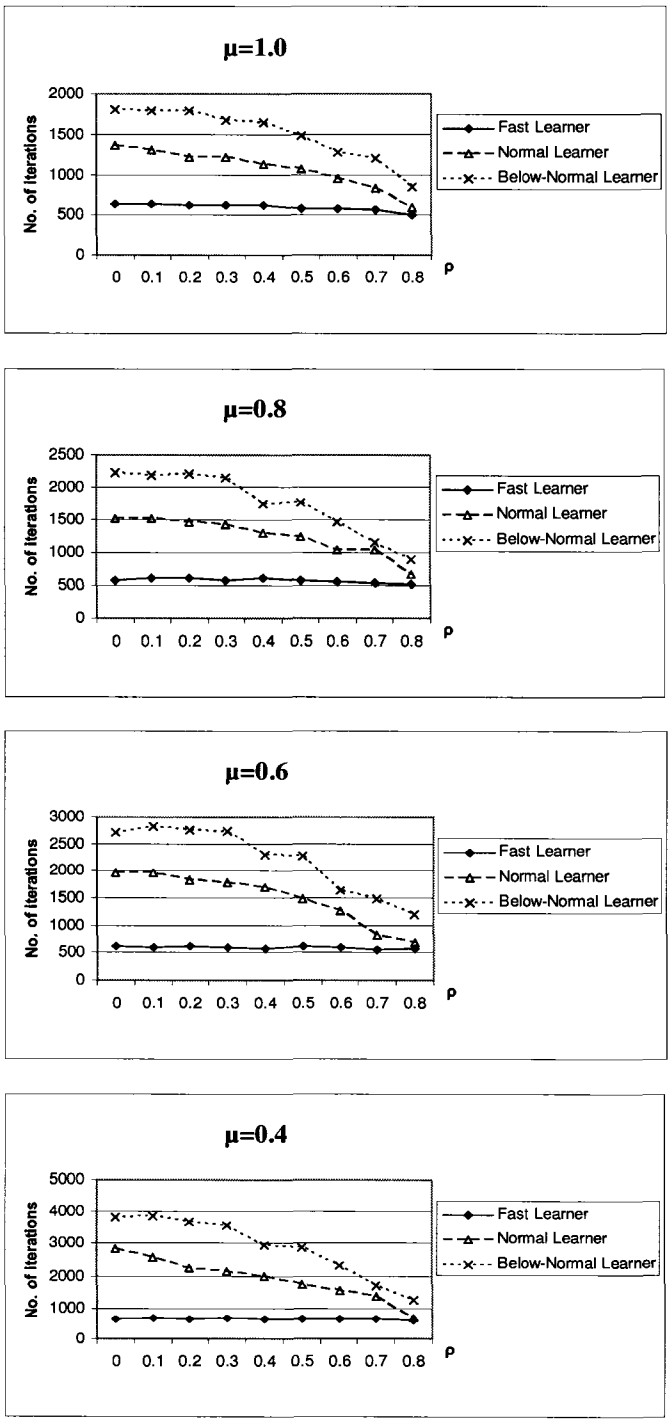

Figure 7.3: The effect of increasing the *Hint Index* provided by the Teacher on Students learning in Environments derived from $E_{10,A}$ by varying the *Difficulty Indices*.

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	0.10	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
1.0	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
	0.0	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
0.8	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
	0.0	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
0.6	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
	0.0	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	0.4	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

Table 7.4: 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 7.2.2).

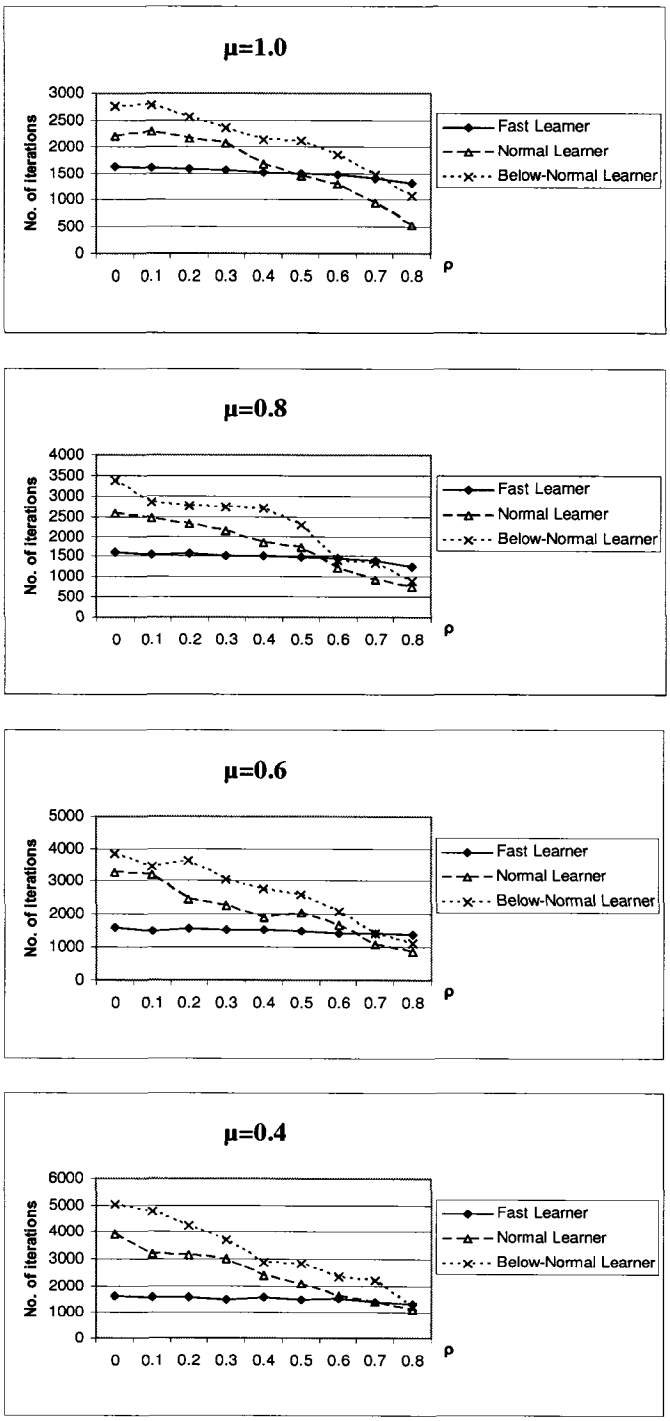

Figure 7.4: The effect of increasing the *Hint Index* provided by the Teacher on Students learning in Environments derived from $E_{10,B}$ by varying the *Difficulty Indices*.

"learn" so as to improve his "teaching" skills, and customize his teaching based on the learning capability of the corresponding Student.

Chapter 8

Modeling the "Learning Process" of the *Teacher*

The basic premise of the research motivating the whole Thesis was to see how we could use a single unified paradigm to model and implement an entire Tutorial-*like* system. While this was achieved in a fairly good manner for all of the other modules discussed in Chapter 3, the crowning achievement would be if this could also be done for the Teacher himself. In other words, the unanswered question is the following: "Can we also incorporate the learning philosophy into the *Teacher* so that he himself could learn a better way of teaching?". This is the goal of this present Chapter.

We have to add, rather apologetically, that the research presented in this direction is far from final. What we include in this Chapter is a *prima facie* case that this is achievable and extremely promising.

The aim of this Chapter is, therefore, to present a new approach to model how the Teacher, in the paradigm of our Tutorial-*like* system, "learns" and improves his "teaching skills" while being himself an integral component of the system. We propose to model the "learning process" of the *Teacher* by using a higher level LA, referred to as the *Meta-Teacher*, whose task is to assist the Teacher himself. Ultimately,

we hope that the latter can communicate the teaching material to the Student(s) in a manner customized to the particular Student's ability and progress. In short, the Teacher, with the help of the *Meta-Teacher* and the input from the *Meta-LA* of Chapter 4, will infer the progress of the Student, and initiate a strategy by which he can "custom-communicate" the material to each individual Student.

We assume that the Teacher is aware of the type (i.e., the "intellectual capability") of each Student. This is communicated to him by the system which uses the *Meta-LA* to infer the learning model of the Student. Although this represents an important component of the information for the Teacher, we argue that it is inadequate. In order for him to customize the teaching experience according to each Student's skill, we propose an LA in its own right, namely the *Meta-Teacher*. The latter will function by intercepting the Student's actions in an intermediate stage of the learning. Based on the Teacher's standards, by which he specifies the required level of assistance for each Student, the *Meta-Teacher* will then infer the required customization that the *Teacher* needs for the particular Student.

In this Chapter, we will first present the structure of the *Meta-Teacher*. We will argue that this entity has to interact with a so-called *Meta-Environment*. We will then demonstrate that our proposed *Meta-Teacher* is successful in improving the Teacher's teaching ability by customizing his teaching according to the "performance" of each Student. Again, by simulating the *Meta-Teacher* within the parameters of the benchmark Environments, we show that our model is successful in determining the level of assistance (specified *via* the *hints*) required for each individual Student so that the latter can appropriately learn more complex material.

We suggest that the results of this Chapter constitute the final piece that makes the puzzle fall into place !

8.1 Introduction

To put it all in the right perspective, we re-iterate the fact that the Teacher in our Tutorial-*like* system is capable of presenting Socratic-type Domain knowledge to the Students. The Domain model contains increasingly complex material that needs to be taught. On the other hand, the Teacher is also capable of providing *hints* to the Students in order to assist them to cope with the complexity and difficulty of the teaching material.

Our Teacher is also aware of the learning model of the Student. He receives this information from the Student model, which, in turn, infers it using the *Meta-LA*, as explained earlier in Chapter 4. Although the Teacher is capable of presenting *hints* to the Students (as explained in Chapter 7), in the context of this present Chapter, we assume that the strength of the *hints* that are to be provided is unknown.

In this Chapter, we propose a novel model for the Teacher to "learn" and improve his "teaching skills" while using the system. This will enable him to "adapt" to the needs and ability of each Student. Thus, as mentioned in the preamble, we introduce the *Meta-Teacher*, which will examine the Student's progress while learning the Domain knowledge. As the name implies, the *Meta-Teacher* uses the LA paradigm as its own internal learning mechanism. It then attempts to infer the incremental amount of *hints* that the Student will need for subsequent learning phases. All of these issues will be clarified in the forthcoming sections.

8.2 Contributions of this Chapter

Based on the above, we state that the salient contributions of this Chapter are as follows:

- The concept of the *Meta-Teacher*, which constitutes a higher-level learning entity attempting to infer the required customization that the *Teacher* needs for

the specific Student.

- The *Meta-Teacher* will make use of the Student model provided by the system. We assume that this model is inferred by the *Meta-LA*, which deduces the learning capabilities of the Student.
- The Teacher will define his *own* standards for each specific Environment, from which the Student is attempting to learn. This will include defining the following aspects:
  - Defining the number of levels for the *hints*. The Teacher can adaptively select one of these levels for each specific Student.
  - For each level, the Teacher needs to define the upper and lower thresholds for the performance of the Students for that level. The details of this will be explained presently.
  - For each of the three Student types (Fast, Normal, and Below-Normal), the Teacher is required to assign the amount of incremental *hints* needed for the specific Student.
- The *Meta-Teacher* will make its inferences based on observing the progress of the Students at an intermediate stage of the learning. The rationale for this will also be explained shortly.
- Based on the knowledge inferred from the *Meta-Teacher* and the Student model, the Teacher will be able to customize the teaching material presented, and provide the appropriate *hints* to each individual Student.

The results from this Chapter demonstrate the *adaptability* of the Teacher to the particular needs and skills of each Student. The results of the experimental simulation, detailed later, performed in different benchmark Environments, show that this model was successful in customizing the teaching to each Student according to the latter's skill and improvement.

8.3 The *Meta-Teacher* Structure

The *Meta-Teacher* is a higher level LA that does not posses a unique distinct Environment of its own. The *structure* of this LA, is in one sense, similar to the structure of the *Meta-LA*, defined earlier in Section 4.4. However, as we shall see, the input, goals, and output are different. In principle, the *Meta-Teacher* represents a synchronous model of interconnected automata.

The structure of the *Meta-Teacher* and *its* associated Environment, which we, in turn, refer to as the *Meta-Environment*, is illustrated in Figure 8.1.

We first explain the principles behind the design and rationale of the *Meta-Teacher*.

The *Meta-Environment* of the *Meta-Teacher* monitors the "progress" of the Student LA over a period of time. The action chosen by the *Meta-Teacher* is the *hint* that the Teacher is providing to the Student. The question now is one of establishing how effective this action is. A simple way of quantifying the performance of the Student is by observing the number of times the Student selects the (unknown) best action that the LA has access to –which is the action with the smallest penalty probability. Based on measuring this performance, the *Meta-Environment* for the *Meta-Teacher* would penalize or reward the action chosen (i.e., the current *hint factor*) of the *Meta-Teacher*. When it concerns evaluating the progress of the Student, we argue that this is best measured at an intermediate stage of the Student's learning process. We believe that this intermediate stage is particularly relevant since at the initial stage, the Student (represented by an LA in a random Environment) will choose the actions in a more random manner. As opposed to this, at the final stages, almost every Student will be close to attaining convergence to a unit vector, and thus to the correct answer.

The *Meta-Teacher* and its *Meta-Environment* incorporate unique features for their interaction model. This can be observed as follows:

1. The *Meta-Teacher* has no real Environment of its own.

2. The *Meta-Environment* for the *Meta-Teacher* consists of observations, which must be perceived as an Environment after doing some "internal" processing of the signals observed.

3. The *Meta-Teacher* has access to the lower level Environment (i.e., the Student's Environment), including the set of the penalty probabilities that are unknown to the Student LA.

4. The *Meta-Teacher* has access to the actions of the lower-level LA, ie., the actions selected by the Student while he is learning.

5. The *Meta-Teacher* has access to the responses made by the lower-level Environment.

6. Finally, the *Meta-Teacher* has access to the Teacher's "standards", in which the latter specifies a mechanism to define the required level of assistance required for each type of Student, and for each Domain of teaching material.

Collectively, items (3)-(6) above represent the *Meta-Environment* for the *Meta-Teacher*. The *Meta-Teacher* (and its companion *Meta-Environment*) are unique in the field of LA.

8.3.1 Formalization of the Teacher's Learning Process Model

In order to simplify our model, we assume that there only four level of *hints*. The Teacher can select one of these levels for each specific Student[1].

The instantiation of the *Meta-Teacher*, which attempts to infer the incremental amount of *hints* that the Student will need for subsequent learning phases, is a 4-tuple: $\{\underline{\alpha}, \underline{\beta}, P, T\}$, where:

$\underline{\alpha} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$, in which

[1]Extending this to consider a wider spectrum of *hint* levels is rather straightforward.

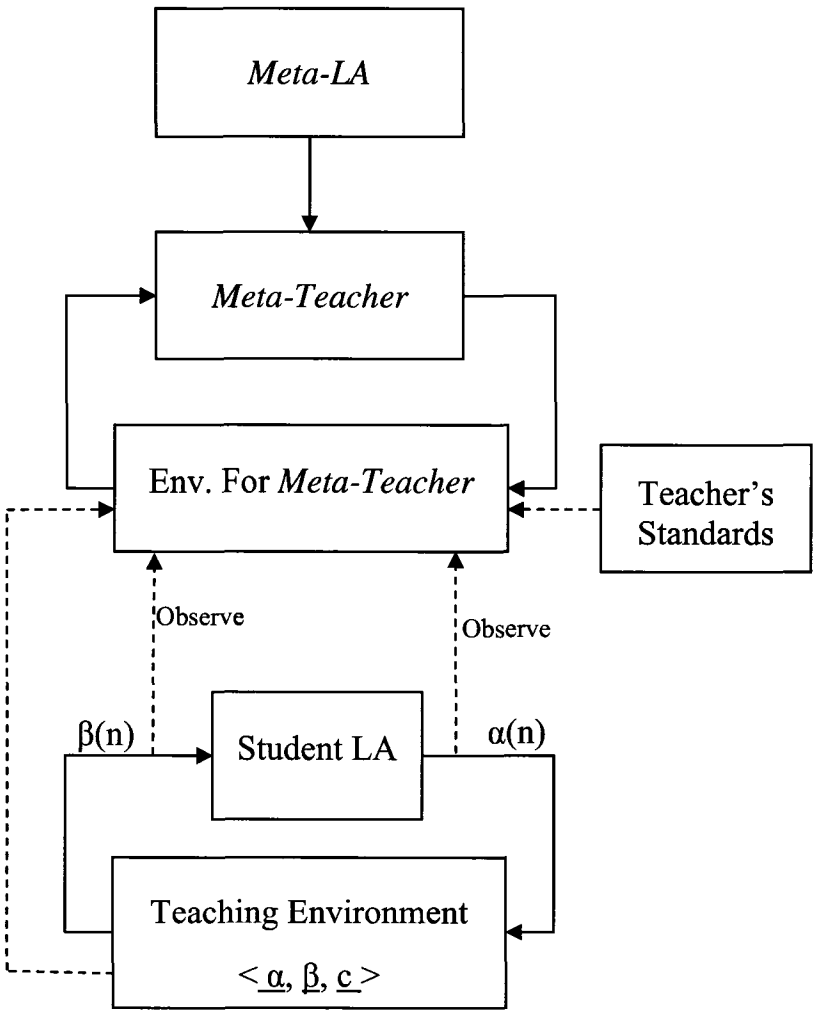

Figure 8.1: Sturcture of the feedback loop between the *Meta-Teacher* and its *Meta-Environment.*

α_i is the action corresponding to the event that the Teacher will select a *hint* of strength level i for the next learning phase. While α_1 represents a *hint* with the least significance, α_4 represents the most suggestive *hints*.

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

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

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

$P = [p_1, p_2, p_3, p_4]^T$, where

$p_i(n)$ is the current probability of the Teacher providing a *hint* represented by α_i.

T is the probability updating rule given as a map as:

T: $(P, \underline{\alpha}, \underline{\beta}) \rightarrow P$.

Each of these will now be clarified.

1. $\underline{\beta}$ is the input which the *Meta-Teacher* receives from its Environment. Let us assume that the current choice of the *Meta-Teacher* is that the *hint* level that the Teacher should use is symbolized by α_i. This means that the *Meta-Teacher* will have to infer a Reward or a Penalty based on how the Student's learning is progressing, and on whether α_i can adequately represent the *hint* to be provided. This, in turn, means that the *Meta-Teacher* must observe the sequence of decisions made by the Student in the different phases of his learning process, and based on this sequence, it must deduce whether its current *hint* level is adequate to the Student or not. We propose to achieve this as follows.

   For a fixed number of queries, which we shall refer to as the "Window", the *Meta-Teacher* assumes that α_i is the *hint* level that leads to the desirable progress of the Student, as depicted in Figure 8.2. Let $\nu(t)$ represent the performance of the Student's learning at time 't'. By inspecting the progress of the Student during

this Window, we would like the *Meta-Environment* to infer whether the *hint* level (i.e., the action the *Meta-Teacher* chooses) is to be rewarded or penalized. The following is the proposed strategy:

- We allow the Teacher to set his standards for the Student's progress so that they are appropriate for the specific level of *hints*.
- For each level of *hints*, the Teacher will set a lower threshold, representing the Student progress in the initial Window, and an upper threshold that represents the Student's progress in the final Window.
- For each level of *hints*, the Teacher will define the incremental *hints* that should be provided to each Student type (Fast, Normal, or Below-Normal), so as to be able to cope with the increasing difficulty/complexity of the next learning phase.
- The progress of the Student is measured during an intermediate stage of learning by observing the number of times the correct action (i.e., action with the least penalty probability for the true Environment) was selected by the Student in the Window.
- If the *current* observed progress of the Student is more close to ν_i, we assert that the *hint* level is best represented by α_i (as shown in Figure 8.2). Now, if the *Meta-Teacher* has chosen α_i, this choice is rewarded. Otherwise, this choice is penalized. For example, in Figure 8.2, at time 't', $\nu(t)$ is best represented by α_3. This implies that at time 't', the *Meta-Environment* will reward the *Meta-Teacher*'s action only if the latter selected action α_3.

2. P is the action probability vector which contains the probabilities that the *Meta-Teacher* assigns to each of *its* actions. At each instant t, the *Meta-Teacher* randomly selects an action $\alpha(t) = \alpha_i$. The probability that the automaton selects action α_i, at time t, is the action probability $p_i(t) = \Pr[\alpha(t) = \alpha_i]$, where $\sum_{i=1}^{r} p_i(t) = 1 \quad \forall\, t$, i=1, 2, 3, 4.

   Initially, the *Meta-Teacher* will have an equal probability for each of the four levels of *hints* as:

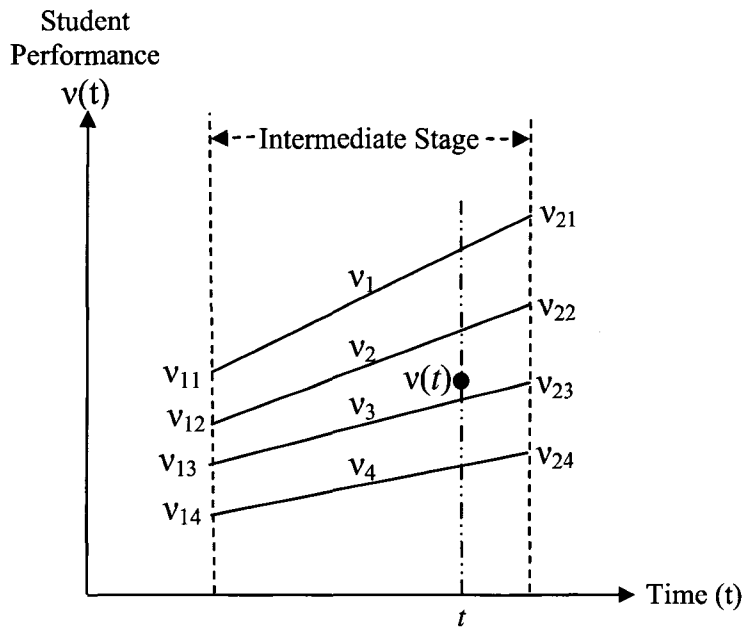

Figure 8.2: Thresholds for the Different Levels of *Hints* and for the Reward/Penalty for the *Meta-Teacher*.

$$[0.25 \quad 0.25 \quad 0.25 \quad 0.25]^{\mathrm{T}}.$$

By observing the sequence of ν(t), the *Meta-Teacher* updates P(t) as per the function T, described presently. By virtue of this interaction, one of the action probabilities, p_j, will, hopefully, converges to a value as close to unity as we want. This convergence indicates that the *Meta-Teacher* has converged to the choice that α_j is the best *hint* level for this Student, and our hypothesis is that if T is chosen appropriately, the strategy will be successful in determining the best *hint* level appropriate for the Student.

3. T is the updating algorithm, or the reinforcement scheme used to update the *Meta-Teacher*'s action probability vector, and is represented by:

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

   This updating algorithm could follow *any* LA learning schemes. For example, if it obeys the DL_{RI} rule, the action probability vector is updated in a *discretized* manner when the *Meta-LA* is rewarded, and is unchanged when the automaton

is penalized.

8.4 Experimental Results

In this Section, we present the experimental results obtained by testing the proposed model for the "learning process" of the *Teacher*. In every case, numerous experiments have been performed to study the improvement of the Teacher's skills while he is teaching the Domain knowledge material to the Students, and how this can affect the level of assistance he provides the Student in their next learning phase.

The Domain knowledge in the simulation contained multiple-choice questions, which were represented in the Domain model by *chapters* with increasing complexity. The simulation was performed against the different benchmark Environments, namely, the two 4-action Environments and the two 10-action Environments, used earlier, and described in Section 4.6.1 and Section 4.6.2. For the different benchmark Environments, again, the simulations were performed with the threshold, T, set to be 0.99, and the number of experiments, NE, = 75. The results of these simulations are described below.

Three types of Students were used in the simulation, and they communicated with the Teacher to learn the Domain knowledge. These three types (i.e., the Fast, Normal, and Below-Normal), were similar to the ones used earlier in Section 5.4.

8.4.1 Teacher's Standards for the Benchmark Environments

One of the components that dictates the behavior of the *Meta-Environment* with respect to the *Meta-Teacher*, is the Teacher's own standards. These standards define the level of *hints* that are associated with every aspect of the Student's progress. As described earlier in Section 8.3, based on the results obtained from an *intermediate* stage of the learning, the Teacher assigns the settings so as to define the Student's

No. of actions in the Env.	*Meta-Teacher* Act.	Learning Threshold				Incremental ρ for Next Chapter		
		Lower Threshold (at 200 iter.)		Upper Threshold (at 600 iter.)		Fast Learner	Normal Learner	Below-Normal Learner
4	α_1	ν_{11}	24	ν_{21}	40	0	0	0
	α_2	ν_{21}	20	ν_{22}	35	0.1	0.1	0.1
	α_3	ν_{31}	16	ν_{23}	30	0.1	0.2	0.2
	α_4	ν_{41}	12	ν_{24}	25	0.2	0.2	0.3
10	α_1	ν_{11}	12	ν_{21}	28	0	0	0
	α_2	ν_{21}	10	ν_{22}	24	0.1	0.1	0.1
	α_3	ν_{31}	8	ν_{23}	20	0.1	0.2	0.2
	α_4	ν_{41}	6	ν_{24}	16	0.2	0.2	0.3

Table 8.1: Teacher's Standards for Classifying the Students' Rate of Progress, for 4-Action and 10-Action Environments.

performance threshold for each of the four levels of *hints*. In our simulation, we defined the intermediate stage as starting from the iteration index 200, and ending with the iteration index 600. We also assigned the 'Window' to be 50 iterations.

In our experiments, these standards have been classified into two groups. The first group defines the standards for the 4-action Environments, while the second group defines it for the 10-action Environments. For each group, and for each Student type, the upper and lower thresholds for the learning progress of the Student were defined. Also, for each *hint* level, the Teacher defined the amount of incremental *hint*, ρ, that had to be provided to the Student in the next phase of learning, as described earlier in Section 8.3.1. These different settings, which were used in the simulations, are tabulated in Table 8.1.

8.4.2 Results using 4-Action and 10-Action Environments

We report now the simulation results for the two sets of benchmark Environments, namely, the two 4-action Environments ($E_{4,A}$ and $E_{4,B}$), and the two 10-action Environments ($E_{10,A}$ and $E_{10,B}$). For each Environment, the simulation was performed for three *chapters* (i.e., for three degrees of difficulty, where μ=1.0, 0.8, and 0.6). Furthermore, for each *chapter*, the simulation was conducted while the Student was currently receiving one of three degrees of *hint factors*, $\rho = 0.0$, 0.2, and 0.4.

The results of these simulations are grouped into two tables. The results for the simulations of the 4-action Environments ($E_{4,A}$, and $E_{4,B}$) are tabulated in Table 8.2, while the results for the 10-action Environments ($E_{10,A}$, and $E_{10,B}$) are shown in Table 8.3.

8.4.2.1 4-Action Environments

In these Environments, the simulation was successful in determining the incremental *hint* level required for the Students in the next learning phase, for all the three types of Students. For the Fast Students, the *Meta-Teacher* always converged to α_1, which, according to the Teacher's standards, translated into a level of "zero" increase of *hints* for the next phase of learning. Using this, the *Meta-Teacher* concluded that the learning progress of the Students was adequate, and that they did not need any additional *hints* for their next phase of learning.

For Normal Students, for most of the *chapters* of these Environments, the *Meta-Teacher* converged to action α_1, which indicated that they did not need any assistance in the subsequent level of learning. However, for the $E_{4,A}$ Environment, when the complexity of the Domain was $\mu = 0.6$, and when the current *hint index* was either $\rho = 0$, or $\rho = 0.2$, the *Meta-Teacher* converged to α_2, which indicated that these Students would be given an incremental *hint factor*, $\rho = 0.1$ (implying that the new *hint factor* would be 0.1 or 0.3 respectively), for their next phase of learning.

As for the $E_{4,B}$ Environment, quite interestingly, the *Meta-Teacher* determined that this represented a more difficult Environment. For *chapters* with complexity μ = 0.8, with either no *hint* or with a *hint index*, ρ = 0.2 given to the Student, the *Meta-Teacher* converged to α_2, leading to the decision of providing an incremental *hint factor* ρ = 0.1 for the Student's next level of learning. For the more complex *chapters*, i.e., μ = 0.6, when the Student was provided with no *hints*, the *Meta-Teacher* converged to α_3 (i.e., suggesting ρ = 0.2 for the next level of learning for the Students). When the Student was provided with *hint index*, ρ = 0.2, the *Meta-Teacher* converged to α_2, concluding that only an additional *hint* of 0.1 (i.e., ρ was now to be set to 0.3) was needed for the subsequent next phase of learning.

On the other hand, the incremental *hints* for the Below-Normal Students were, in some cases, more than those required for the Normal Students. For example, in the $E_{4,A}$ Environment, for *chapters* with complexity, μ = 0.6 and with no *hint* given to the Student, the *Meta-Teacher* converged to α_3, indicating that the suggested level of *hint* in the next learning phase is 0.2. Even for the complexity just mentioned, and with existing *hints* given to the Student to be of values ρ = 0.2 or 0.4, the *Meta-Teacher* converged to α_2, suggesting an incremental *hint index* ρ = 0.1.

8.4.2.2 10-Action Environments

Although the 10-action Environments are considered to be more difficult than the 4-action Environments, the results obtained were analogous to those of the former. The simulation was successful in determining the *hint* level required for the Students for their subsequent learning phases. This was true for all the three types of Students described earlier.

Again, for the Fast Students, the *Meta-Teacher* always converged to α_1, which translated into a level of "zero" incremental *hints* required for the next learning phase. Thus, Fast Students proved, again, that they do not need any additional assistance from the Teacher.

Env.	μ	ρ	Incremental ρ for Next Chapter					
			Fast Learner		Normal Learner		Below-Normal Learner	
			α	incr. ρ	α	incr. ρ	α	incr. ρ
$\mathbf{E_{4,A}}$	1.0	0.0	α_1	0	α_1	0	α_1	0
		0.2	α_1	0	α_1	0	α_1	0
		0.4	α_1	0	α_1	0	α_1	0
	0.8	0.0	α_1	0	α_1	0	α_2	0.1
		0.2	α_1	0	α_1	0	α_2	0.1
		0.4	α_1	0	α_1	0	α_1	0
	0.6	0.0	α_1	0	α_2	0.1	α_3	0.2
		0.2	α_1	0	α_2	0.1	α_2	0.1
		0.4	α_1	0	α_1	0	α_2	0.1
$\mathbf{E_{4,B}}$	1.0	0.0	α_1	0	α_1	0	α_2	0.1
		0.2	α_1	0	α_1	0	α_2	0.1
		0.4	α_1	0	α_1	0	α_1	0
	0.8	0.0	α_1	0	α_2	0.1	α_2	0.1
		0.2	α_1	0	α_2	0.1	α_2	0.1
		0.4	α_1	0	α_1	0	α_1	0
	0.6	0.0	α_1	0	α_3	0.2	α_3	0.2
		0.2	α_1	0	α_2	0.1	α_2	0.1
		0.4	α_1	0	α_1	0	α_2	0.1

Table 8.2: *Meta-Teacher*'s Learning Process in Benchmark 4-Action Environments.

For Normal Students, and for most of the *chapters* of these Environments, the *Meta-Teacher* concluded that they do not need any assistance in the subsequent learning phases; it converged to α_1 in these cases. In a few instances, Normal Students required additional incremental *hints* for the next learning phase. For example, in the $E_{10,A}$ Environment when the complexity of the Domain was $\mu = 0.8$, and no *hints* were currently given, the *Meta-Teacher* converged to α_2, which suggested an incremental *hint factor* $\rho = 0.1$. Only in the more difficult *chapters*, when μ was 0.6, and with no *hint* provided to the Student, did the *Meta-Teacher* converge to α_3, which indicated that the Student needed a *hint* level with an incremental value of $\rho = 0.2$ for the next level of learning.

Finally, we report the case for Below-Normal Students. They, on the other hand, required, in some cases, more *hints* than those of Normal Students –as would be expected. For example, for both the $E_{10,A}$ and $E_{10,B}$ Environments, in *chapters* with complexity $\mu = 0.6$ and with no *hint* given to the Student, the *Meta-Teacher* converged to α_4, which suggests that these Student, in their next learning phase, should receive a *hint* with value $\rho = 0.3$.

As the reader will observe, the results reported are quite brief. But they represent the typical behavior of the *Meta-Teacher* in these complex learning problems.

8.5 Conclusions

This Chapter answered a basic, but a crucial question that was presented in the preamble of this Chapter, namely that of determining whether we could incorporate the fundamental LA philosophy into the *Teacher* so that he himself would learn a better way of teaching. We demonstrated here that the answer to this question is in the affirmative. Although the model which we used incorporated these concepts in fairly simplistic manner, we believe that it proves that the concept is achievable, valid, and very promising.

Env.	μ	ρ	Incremental ρ for Next Chapter					
			Fast Learner		Normal Learner		Below-Normal Learner	
			α	incr. ρ	α	incr. ρ	α	incr. ρ
$\mathbf{E_{10,A}}$	1.0	0.0	α_1	0	α_1	0	α_2	0.1
		0.2	α_1	0	α_1	0	α_1	0
		0.4	α_1	0	α_1	0	α_1	0
	0.8	0.0	α_1	0	α_2	0.1	α_3	0.2
		0.2	α_1	0	α_1	0	α_2	0.1
		0.4	α_1	0	α_1	0	α_1	0
	0.6	0.0	α_1	0	α_3	0.2	α_4	0.3
		0.2	α_1	0	α_2	0.1	α_3	0.2
		0.4	α_1	0	α_1	0	α_1	0
$\mathbf{E_{10,B}}$	1.0	0.0	α_1	0	α_1	0	α_2	0.1
		0.2	α_1	0	α_1	0	α_1	0
		0.4	α_1	0	α_1	0	α_1	0
	0.8	0.0	α_1	0	α_2	0.1	α_3	0.2
		0.2	α_1	0	α_1	0	α_3	0.2
		0.4	α_1	0	α_1	0	α_1	0
	0.6	0.0	α_1	0	α_3	0.2	α_4	0.3
		0.2	α_1	0	α_2	0.1	α_3	0.2
		0.4	α_1	0	α_1	0	α_1	0

Table 8.3: *Meta-Teacher*'s Learning Process in Benchmark 10-Action Environments.

In this Chapter, we presented a new approach to model how the Teacher can learn and improve his own teaching skills within our learning paradigm. We proposed to model this learning process by means of a higher level LA (the *Meta-Teacher*) whose task is to assist the Teacher to try to identify the level of assistance that he would provide to the Student in their next learning phase. Using our model, we demonstrated that the Teacher can communicate the teaching material to the Student(s) in a manner customized to the particular Student's ability and progress.

As a "proof of pudding", in the next Chapter, we will present our implementation of the entire Tutorial-*like* system. We will present the User Interface of the prototype, and present some simulation results of the Students' performance during these simulations.

Chapter 9

LAB-ITS: A Prototype LA-Based Intelligent Tutorial-*like* System

This Chapter presents LAB-ITS, the prototype implementation of our **LA-B**ased **I**ntelligent **T**utorial-*like* **S**ystem. This prototype incorporates all the aspects that we researched and proposed in the preceding Chapters of this Thesis, i.e., from Chapter 3 to Chapter 8. The intention is that the prototype in its entirety, will function to simulate a Classroom of Students that need to learn the Domain model. This includes the interactions of these Students with the Teacher, the inter-Student "discussions", and the *hints* provided by the Teacher. The prototype will simulate the learning of the Students, and examine their respective progress during the teaching/learning sessions.

9.1 Introduction

The goal of the prototype is to provide a researcher with the software necessary to simulate the teaching/learning experience within the generalized framework proposed by the tenets of the Thesis (as motivated in Chapter 3). In the interest of conciseness,

the salient aspects of each of the relevant modules is not reiterated here.

In LAB-ITS, we simulate how Students learn and interact with the Teacher and with each other. The simulation is capable of testing any reasonable hypotheses for the learning of the Students. For example, a researcher can test these two hypotheses:

- How can we compare the learning of 2 groups, where the first group has 8 colleagues all being Normal learners, and the second group has 5 Superior Students and 3 Weak Students.
- Is a weak learner better-off learning with another set of (a) 8 colleagues (3 Fast, 3 Normal, and 2 Slow learners), where he completely transfers knowledge from them, or (b) 4 Fast colleagues, where he partially utilizes their knowledge.

The Chapter will briefly describe[1] the design of the system, and show the sequence of tasks that the researcher encounters in order to run the simulation, and view/examine the results. In particular, we submit various screenshots of the different User Interface (UI) screens so as to illustrate the applicability and power of the prototype.

The different components of LAB-ITS were tested using a bottom-up philosophy, where each module was tested and verified independently. However, when incorporating these modules together, LAB-ITS offers numerous different configuration parameters permitting a combinatorial explosion of simulation scenarios. In our testing of LAB-ITS, we have only tested a few "reasonable" configuration parameters, and they seem to verify that the package functions properly.

We close this introductory section by mentioning that a fully functional version of LAB-ITS is available and can be demonstrated on request.

[1]Since this is a Doctoral thesis, the programming-related implementation details of the package are omitted. However, they can be made available to researchers who would like to collaborate with us.

9.2 Top-Level Concepts: LAB-ITS

9.2.1 Main Screen

The overall applicability of the LAB-ITS system involves three essential phases. In these phases, the user can:

1. Configure the prototype and specify the parameters of each of the associated modules. The details of this configuration phase are explained in the next few sections.
2. Run the simulation experiments, as per the parameters defined.
3. Graphically present the results of the overall Teaching/Learning session.

When the researcher enters LAB-ITS, he will view the main screen of the prototype, depicted in Figure 9.1. The screen displays the different steps that are to be performed from the prototype's main portal. This includes the following:

- Specifying the characteristics of the Students/*Classroom*, including the Students' identities, learning types, interaction strategies, and knowledge exchange strategies.
- Defining the characteristics of the Teacher, and the range of the *hint index*, ρ, that he provides to the Students.
- Providing the configurations for the Domain, the characteristics of the Environment and the range of the difficulty index, μ.
- Specifying the parameters for the *Meta-LA* and its threshold values.
- Providing the *Tactic-LA* parameters, including the values for its Environment's reward probabilities.

- Defining the experimental configurations, either being undertaken as a single run or an ensemble. Also, if the experiment is conducted over a single chapter or multiple chapters.

The main screen also permits the "Run" and "Display" modes of LAB-ITS.

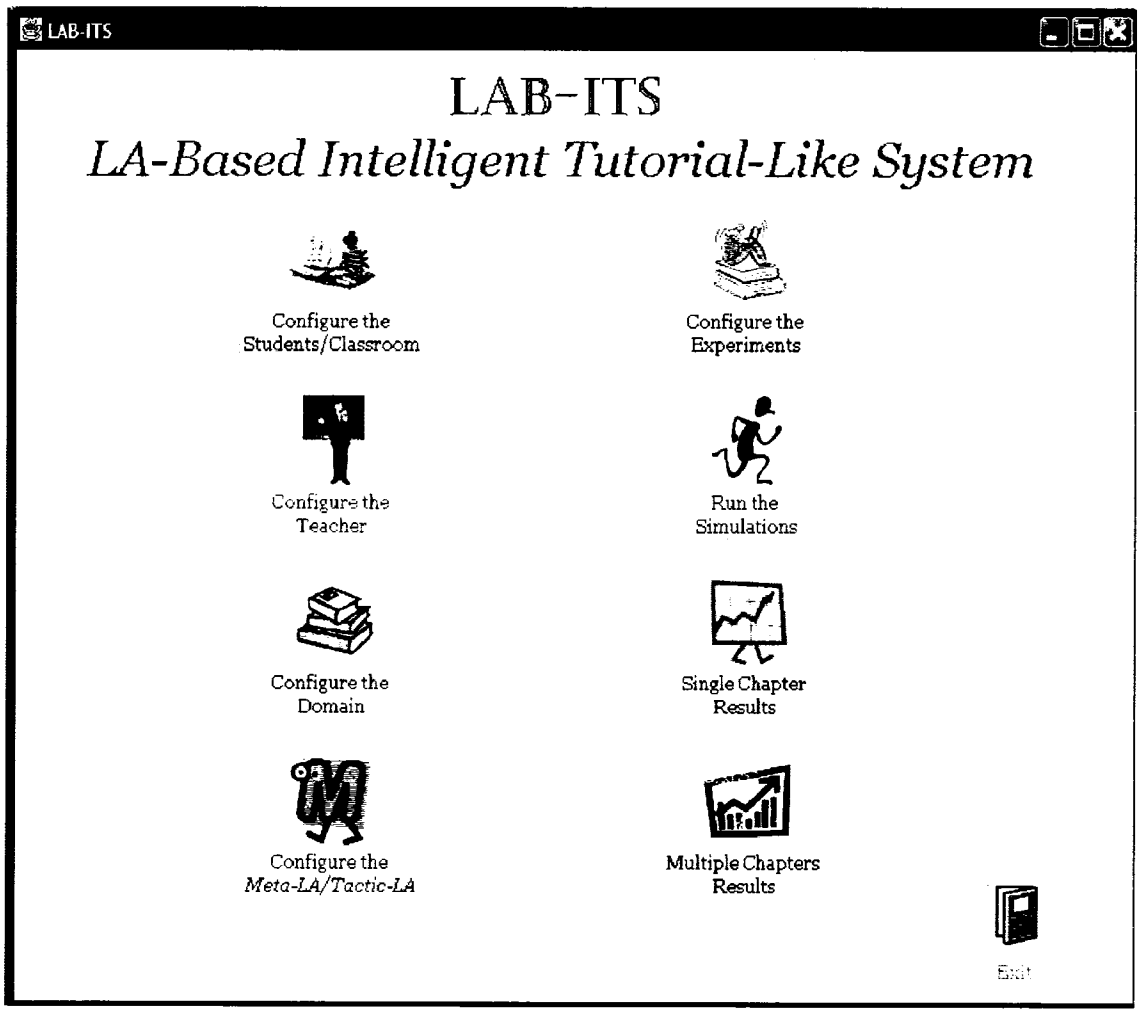

Figure 9.1: LAB-ITS Main Screen.

9.3 Configuration of the System

9.3.1 Configuring the Students/Classroom

First of all, the researcher needs to configure the Student Simulators that mimic the Students who are learning the Domain knowledge. The screenshot of the Students' configuration is shown in Figure 9.2.

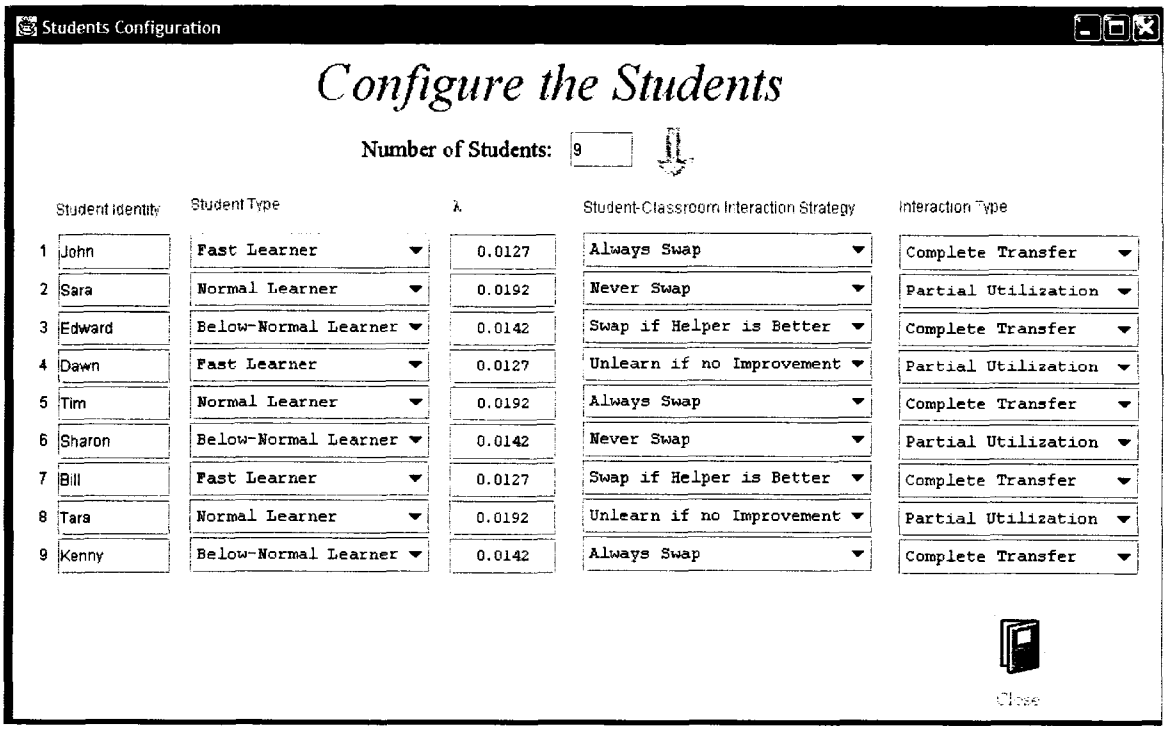

Figure 9.2: The Screenshot for Configuring the Students/Classroom.

On entry to the system, the researcher needs to define the number of Students who are participating in the simulation experiments, and the characteristics of each Student. This includes specifying the following:

- The identity of the Students. These are the names (or identification "tags") of each 'Entity' or component that is involved in the Teaching/Learning exercise.

In the interest of simplicity, the displayed screenshot specifies them using the names of individuals. This identity is displayed as the first attribute of the screen.

- The Students' learning type. The prototype permits each Student to be one of three following types:
  - A Fast learner, represented using a Pursuit scheme, as per Section 5.4.
  - A Normal learner, mimicked by a VSSA, as per Section 5.4.
  - A Below-Normal learner, simulated by a VSSA, as per Section 5.4.

  The classification of the respective Students is given as the second attribute in the screenshot.

- The parameter characterizing each respective Student LA. This involves setting the learning parameter of the underlying LA. For each Student type, the system has a default λ, and this value can be modified by clicking on it. This would take the researcher to a new screen with a scrollbar (see Figure 9.3), which gives him the freedom to modify it. We mention that the range of values permitted by the scrollbar is within the range suitable for the respective type of LA. This is the third attribute of the current screen.

- The Student-Classroom interaction strategy. This is specified as the fourth attribute of the screen and assumes one of the four possible values:
  - **Always Swap**: The Student always considers the knowledge of other Students as being credible.
  - **Never Swap**: The Student works independently and chooses not to communicate with other fellow Students.
  - **Swap if Helper is Better**: The Student evaluates the knowledge received from the other Students, and accepts this knowledge only if he can infer that this information is credible.

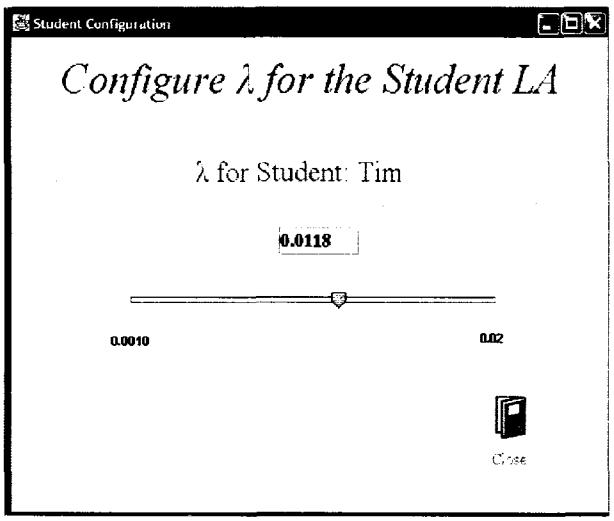

Figure 9.3: The Screenshot for Modifying the λ of the Student Simulator's LA.

- **Unlearn if no Improvement**: The Student evaluates the knowledge received from the other Students. Subsequently, after a period of probation, he "unlearns" the information gleaned if he infers that this information was misleading.

- The interaction type. This attribute, which is the last attribute in the screenshot, can take the form of either:
  - Complete transfer of information: The Student will consider all the knowledge from the other Student.
  - Partial utilization of information: The Student will use the knowledge from the other Student(s) only to "improve" his knowledge.

9.3.2 Configuring the Teacher

The next aspect of the configuration module requires the researcher to configure the Teacher model. This is done by invoking the "Configure the Teacher" icon in the main screen, which brings up the screen shown in Figure 9.4. The actual configuration will

be achieved by defining the respective values of the *hints factor*, ρ, which the Teacher utilizes to provide *hints* to the Student. These *hints* assist them in the learning process, as described earlier in Chapter 7. Defining this quantity requires setting the values of ρ such as its lower threshold (the "Lower Threshold" value in Figure 9.4), the upper threshold (the "Upper Threshold" value in Figure 9.4), and the "Resolution" or the step size. These values can be assigned in either of two ways:

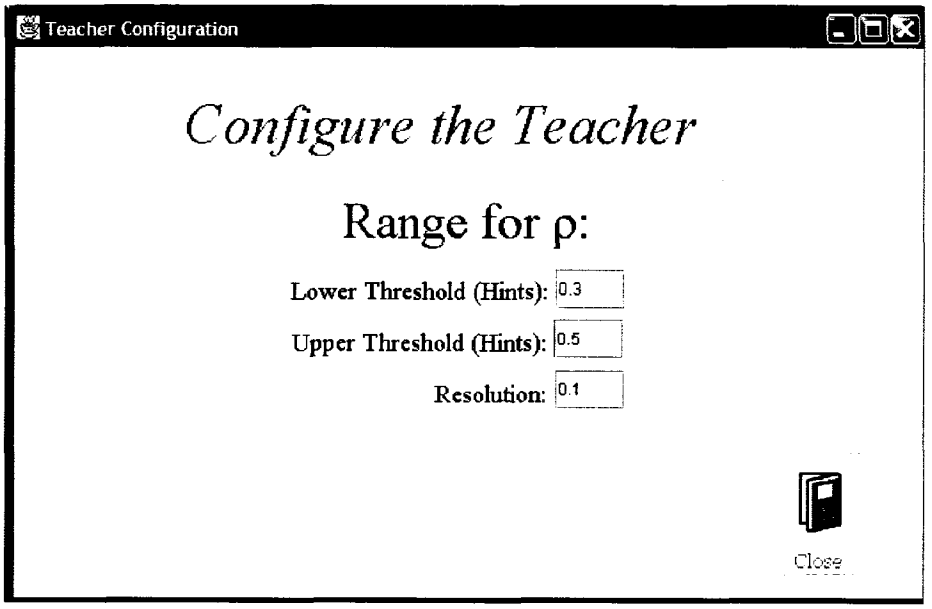

Figure 9.4: The Screenshot for Configuring the Teacher.

- To simulate the teaching of a *single* chapter. This setting permits only a single value of μ in the Domain configuration. In this case, the Teacher will try to teach the Students the contents of the specified chapter, and this will be done by providing *hints* with different values of ρ, as shown in the configuration screen.
- To simulate the teaching of *multiple* chapters. When ρ is assigned a specific value, the Teacher will use *this* value to teach multiple chapters of the Domain knowledge. On the other hand, if ρ is described by a range of values, the Teacher will use a different *hint factor*, ρ, for each chapter in the Domain knowledge. This requires the values of the *hint indexes* (defined in the configuration setting

of the Teacher) to be aligned with the *difficulty indexes* given in the Domain configuration. If these do not match, the system provides a "warning" to the user.

9.3.3 Configuring the Domain

In this screen, as shown in Figure 9.5, the researcher is given the ability to define the Domain knowledge. This includes defining two sets of values. The first describes the Environment, from which the Student is trying to learn. The second defines the *difficulty indexes*, μ, which describes the different chapters of the Domain knowledge, with increasing complexity.

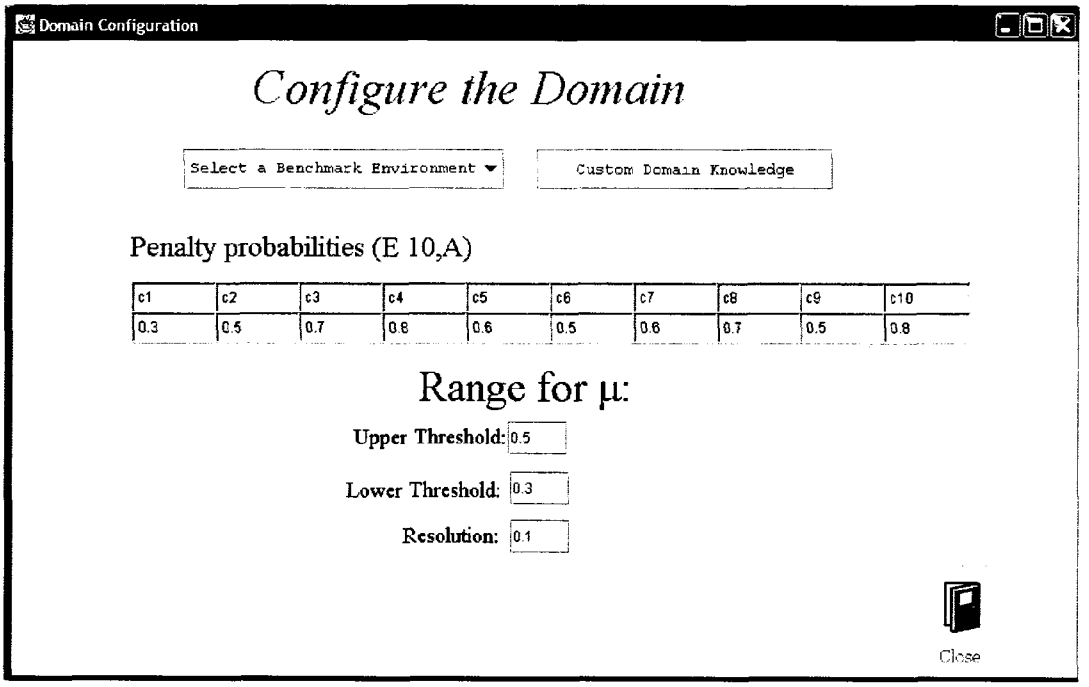

Figure 9.5: The Screenshot for Configuring the Domain.

To define the Environment, the researcher is given the option of resorting to one of the existing benchmark Environments ($E_{4,A}$, $E_{4,B}$, $E_{10,A}$, or $E_{10,B}$), described earlier in Section 4.6.1 and 4.6.2. The reader will observe that these Environments have been

used in the different Chapters of the Thesis. Alternatively, the researcher can devise a custom-defined Environment by clicking on the "Custom Domain Knowledge" icon. On choosing this option, the researcher will be provided with a fresh screen, as shown in Figure 9.6, where he will be permitted to set:

- The number of actions in the Environment.
- The penalty probability, c_i, for each of these actions.

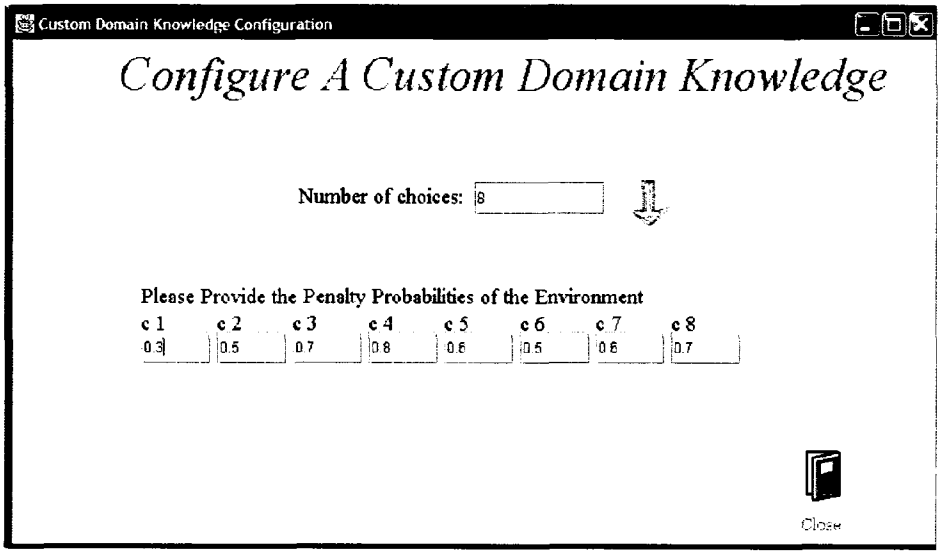

Figure 9.6: The Screenshot for Configuring a Custom Domain Knowledge.

The second set of parameters that have to be assigned so as to adequately configure the Domain, pertain to the chapters of the Domain, quantified in terms of the *difficulty factor*, μ. The researcher is required to set the values of μ, i.e., the "Upper Threshold", the "Lower Threshold", and the "Resolution" or the step size. These values can be seen in Figure 9.5.

Observe that a more complex Environment now requires a larger *hint factor*, as per the reasoning of Chapter 7.

9.3.4 Configuring the *Meta-LA/Tactic-LA*

In order to define the characteristics of the Student Modeler, as defined in Chapter 4, the parameters of the *Meta-LA* need to be set. Also, to describe the interaction decisions of the Student with the Classroom, the settings of the *Tactic-LA* needs to be specified, as mentioned in Chapter 5. The screen for assigning these parameters is shown in Figure 9.7.

For the *Meta-LA*, the researcher needs to set the lower and upper bounds of θ (θ_1 and θ_2), as defined in Chapter 4. These specify the upper and lower thresholds for the rates of learning for the corresponding Students.

With regard to the *Tactic-LA*, the researcher needs to first define the number of iteration cycles during which the Student must attain a decision as to whether he chooses to communicate with others in the Classroom or not. Thereafter, the reward probabilities of the *Tactic-LA*'s Environment needs to be initialized, as defined earlier in Section 5.3.1.1, which is also done in the present screen.

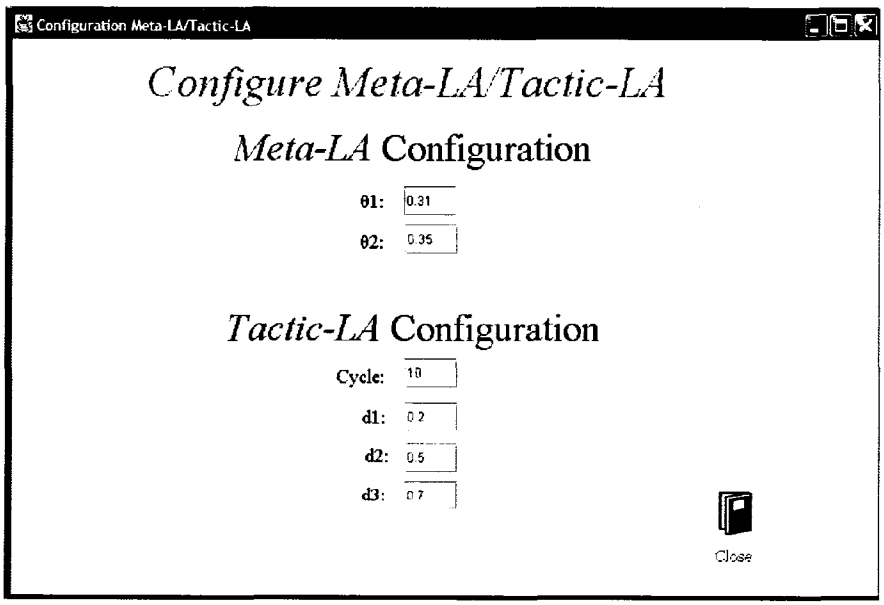

Figure 9.7: The Screenshot for Configuring the *Meta-LA/Tactic-LA*.

9.3.5 Configuring the Experiments

The simulation of the Tutorial-*like* system can be executed either for a single run, or for an ensemble of experiments. The researcher has the freedom to specify the type of execution mode in which he is interested. If he is interested in an ensemble of experiments, he has to then specify the number of experiments. The default option is to execute the simulation for NE = 75 experiments.

In an analogous manner, the researcher has to also specify if the experiments are to be done for a *single* chapter (with multiple values of the *hint index*, ρ), or for *multiple* chapters. If it is for a single chapter, the system should have only a single setting for μ in the Domain configuration (see Section 9.3.3).

The screenshot for this configuration can be seen in Figure 9.8.

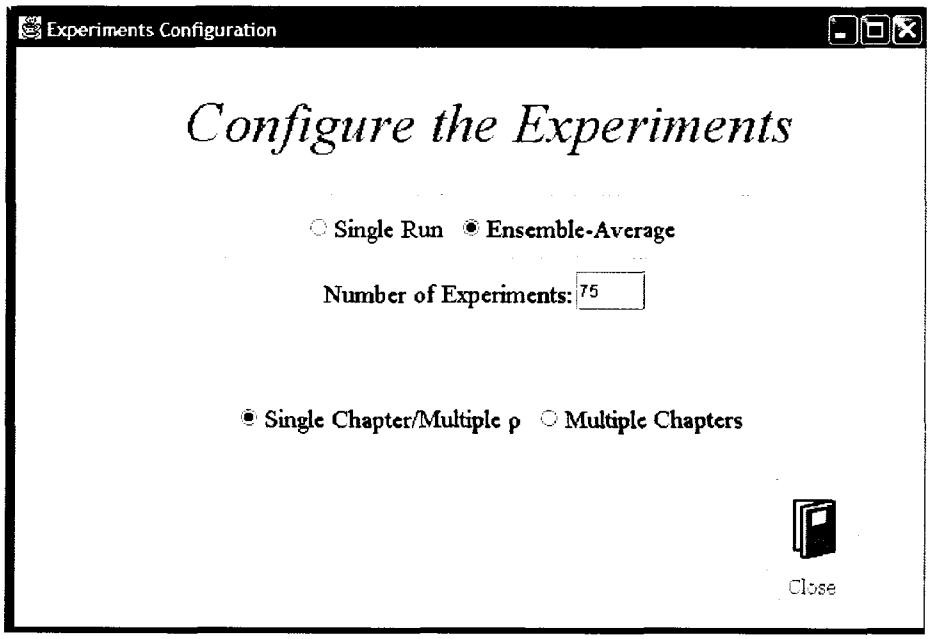

Figure 9.8: The Screenshot for Configuring the Experiments.

9.4 Running the Simulation Experiments

The second stage of executing the prototype involves *running* the simulation experiments. After the above configurations have been set, the researcher can click on the "Run the Simulations" icon. Upon doing this, the system achieves an automatic verification of the parameters of the configurations, so as to ensure that they are not contradictory. If there are any discrepancies in the parameters, the system will display error messages so that the researcher can correct them before proceeding.

An example of a mismatched configuration would be that of defining a range for ρ (in the screen which assigns the Teacher's configuration) which does not match the corresponding range for μ in the Domain configuration. Other errors occur if the researcher requests a "Single Chapter" in the configuration setting for the experiments (which implies that there should be only a single value for μ), while specifying a range of values for μ in the screen that configures the Domain.

The simulation continues if no discrepancies are encountered. The time required to run the simulation may vary, depending on the configuration parameters, especially, the number of experiments and the number of Students.

9.5 Displaying the Simulations Results

The final stage of executing LAB-ITS involves presenting the results of the simulations. These results are shown graphically, and can depict the progress of the Students who participated in the simulation. This display can be one of two types as detailed below.

9.5.1 Single Chapter Results

If the researcher requests that the simulation be executed for a single chapter (and appropriately defined a range for the *hint factor*, ρ, in the screen defining the configuration of the Teacher), he needs to select the option of "Single Chapter Results" in the main screen. Under this option, the Students learn the contents of the Domain knowledge of a single chapter (where the complexity does not change) multiple times. In each invocation, the Teacher teaches the Domain knowledge with an increasing value of ρ, which was earlier specified in the Teacher's configuration. A typical screenshot of the results obtained, is depicted in Figure 9.9.

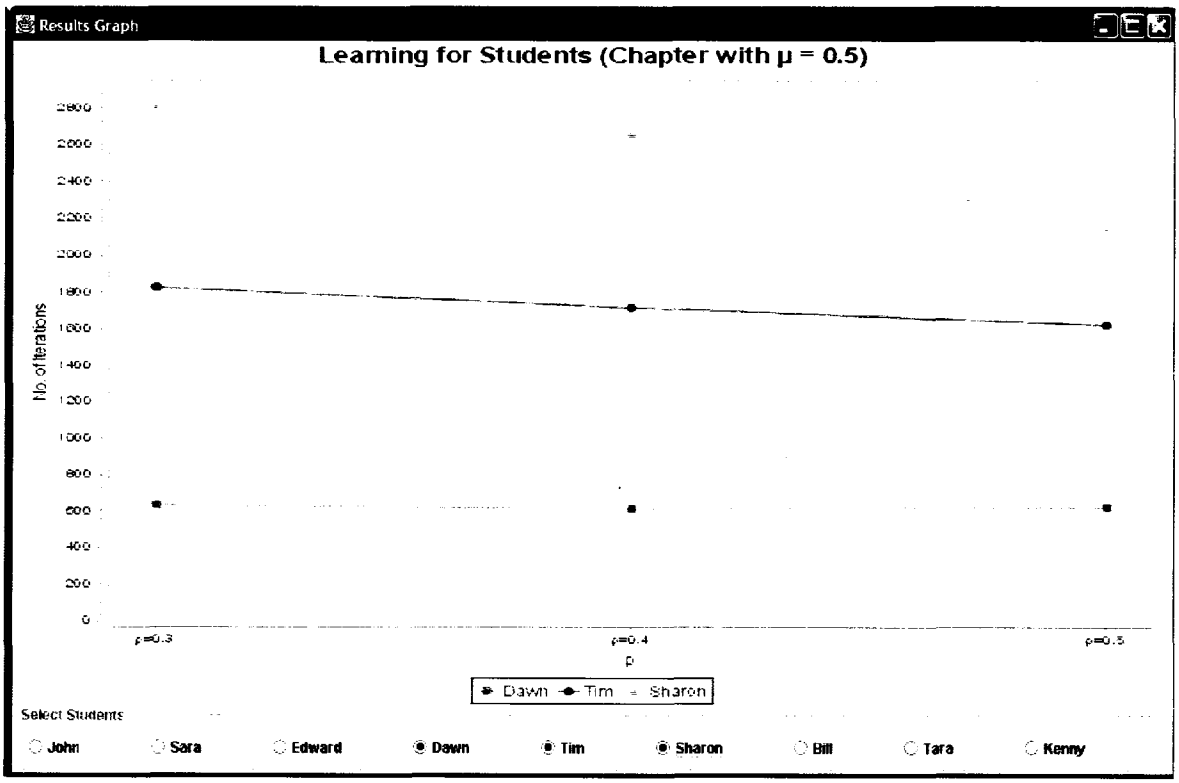

Figure 9.9: The Screenshot for Viewing the Results of a Single Chapter.

The chart depicts the learning of the Students for a single chapter where:

- The X-axis represents the value of the *hint factor* ρ, which is provided by the

Teacher in each experiment.

- The Y-axis represents the number of iterations needed for the Student to learn the specific chapter.

The screen also specifies a list of all the Students who have participated in the experiments. To render the system more user-friendly, clicking on a specific Student will include the results pertinent to *this* Student in the graph[2]. This mode of displaying the results enables the researcher to compare the learning capabilities of one or more Students simultaneously. For example, Figure 9.9 displays the results of three Students, named symbolically as Dawn, Tim, and Sharon.

9.5.2 Multiple Chapters Results

When the researcher selects the "Multiple Chapters" option in the screen that configures the experiments, the Domain model will be presented to the Students *via* chapters with increasing difficulty. The latter will be defined by the range of the *difficulty index*, μ. The screenshot of this mode is shown in Figure 9.10. The prototype then presents the results of how the Students learned the teaching material as follows:

- The X-axis specifies the *complexity index*, μ, and the corresponding *hint factor*, ρ, which was used for the relevant chapters.
- The Y-axis presents the number of iterations needed by *each* Student to learn the chapter.

Again, the list of all Students participating in the simulations are listed at the bottom. This permits the researcher to be able to include or remove one or more Students in the display. This allows for a comparative and objective quantification of the Students' learning abilities.

[2]A subsequent click will cause the Student's results to disappear from the graph.

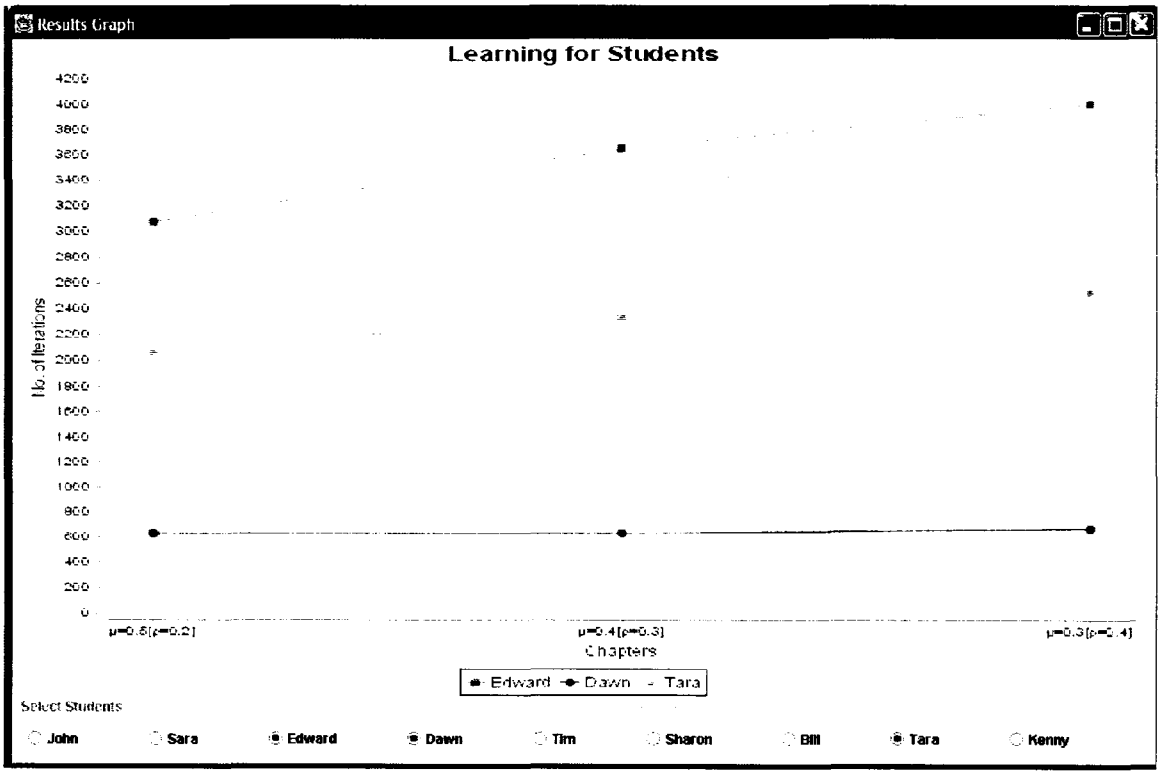

Figure 9.10: The Screenshot for Viewing the Results of Multiple Chapters.

9.6 Conclusion

In this Chapter, we have presented LAB-ITS, a LA-Based Intelligent Tutorial-*like* system prototype. The aim of the Chapter was to provide the reader with insights of how the prototype functioned. We presented the sequence of tasks that a researcher would encounter when using the system. We also submitted various screenshots of the different UI screens to illustrate the capabilities and potential of the prototype.

In particular, the Chapter also illustrated how a researcher could configure the Students/Classroom, their model of interaction strategy, the Teacher and the *hints* he provided, the Domain and its complexity, and the *Meta-LA* and *Tactic-LA* parameters. Finally, the Chapter included the experimental configuration, and details of how to execute and view the results of the simulations.

Chapter 10

Conclusions and Future Work

The goal of this research project was to study, design and implement systems that could tutor other systems using techniques that traditional *real-life* Teachers use where they teach *real-life* Students. Our intention was, however, to include in this general set-up the scenario where the system representing the Teacher was imperfect, the Domain was Socratic, the Students were, possibly, cooperating between themselves, and the Teacher was himself "teachable". We believe that we have succeeded, at least from a prototype point of view, in this endeavor.

This Chapter summarizes the results that we have obtained in this regard, in a Chapter-by-Chapter basis. Also, rather than list the future work as a separate section, we report the avenues for future research in each section, as is relevant to each module.

10.1 Overall Contributions

The results introduced in this Thesis reported a new class of systems, which merged the fields of Intelligent Tutorial Systems (ITS) and Learning Automata (LA), which

we refer to as the *primary* areas. We refer to these systems as being Intelligent Tutorial-*like* systems. Our research has succeeded in developing such Tutorial-*like* systems where every module was composed using LA, and where *every* entity improved its behavior based on the response it received from *its* corresponding Environment.

Our Tutorial-*like* system has incorporated different concepts that are novel and unique to either or both the above mentioned *primary* areas. In our system, every facet of the interaction involved a model (or a non *real-life* Student), and in which the design and implementation followed a formal established learning paradigm. Every module of the system utilized the fundamental principles of LA. First of all, the Student (i.e., the learning mechanism) was modeled using a LA. The *imperfect* Teacher was also modeled using an LA Environment. The Classroom of Students was also modeled using LA, with each of them being represented by *different* types of LA.

Throughout the Thesis, we argued that LA-based Intelligent Tutorial-*like* systems can serve to be an easy and useful way to model and implement system-based (as opposed to *real-life* Student-based) Tutorial systems. It is our belief that after our problem has been satisfactorily solved within a machine learning perspective, it can be, hopefully, ported to the application domain of Intelligent Tutorial systems.

The Thesis presented a philosophic view of such Tutorial-*like* systems in which we relaxed the infallible constraint that the "Teacher is 'perfect' ". By modeling the Teacher as a stochastic "Entity", we presented a valid type of Teacher, who is, in fact, an Environment or Medium in which the Student is learning. Such a generalized Teacher can also resemble a scenario applicable for traditional Tutorial systems.

We now present our contribution of the Thesis in a module-by-module manner.

10.2 Model of the Student

Our first contribution involved presenting a novel strategy for *inferring* the learning model of a Student, or a Student Simulator, in a Tutorial-*like* system. The Student Modeler itself used LA as an internal mechanism to determine the learning model of the Student. This was intended to enable the Tutorial-*like* system to customize the learning experience for each Student. To achieve this, we proposed that the Student Modeler use a higher-level automaton, a so-called *Meta-LA*, which observed the Student Simulator's actions and the teaching Environment, and attempted to characterize the former's learning model. The *Meta-LA* tried to determine if the Student in question was a fast, normal, or slow learner.

In Chapter 4, we demonstrated the validity of the *Meta-LA* concept as a modeler for the Student. Additionally, the Chapter presented a new scheme of interconnecting automata, where the behaviorial sequence of one automaton affects another. The other distinct differences of such an interconnection of LA are listed in the Chapter.

10.2.1 Future Work

Apart from the fundamental novelty of the approach, our proposed Student Modeler has additional advantages. Indeed, it demonstrates the ease of design, implementation, and testing using a LA paradigm. We believe that our approach can be ported for use in traditional Tutorial Systems – which is a problem open for future research. Also, by determining a method to *measure* the rate of learning for *real-life* Students, the Student Modeler should be able to "approximate" their respective learning models. This would also help to decrease the complexity of implementing Student models in traditional Tutorial Systems.

10.3 Model for the Student-Classroom Interaction

With regard to the research in this area, in this Thesis we tackled a new problem that has not been considered before in the literature pertaining to Tutorial systems. This involves the setting where a single Teacher is allowed to teach multiple *communicating* Students of varying calibers.

In Chapter 5, we presented a novel approach to model the Student-Classroom interaction. Apart from the conceptual contribution, the model permitted the Students to communicate with each other (if they chose to), thus enabling them to learn not only from the Teacher, but also from other colleague Students in the Classroom.

We also provided the model with the potential of Students deciding on their intended interaction strategy with other Students. This gave them the control to also work independently. The Chapter involves a novel LA interaction which, when incorporated into the domain of Intelligent Tutorial systems, appears to be unreported in either of the so-called *primary* areas.

10.3.1 Future Work

We believe that our approach can also be ported for use in traditional Tutorial Systems. By allowing *real-life* Students to interact with each other, in addition to their interaction with the Teacher, we believe that their learning can be improved. This would enable traditional Tutorial systems to be more efficient and realistic.

Integrating these concepts into *real-life* Tutorial systems is an avenue for future research.

10.4 Modeling an Increasingly Complex Domain

In the research represented by this Thesis, we presented a novel approach to progressively increasing the complexity of an Environment, in a LA paradigm. Our model allowed the Tutorial-*like* system to increase the compexity/difficulty of Socratic-type Domain knowledge, in which the more complex knowledge was taught in subsequent chapters. This was rendered possible by correspondingly reducing the range of the penalty probabilities of all actions by incorporating a scaling factor, μ. The system enabled the Students to learn an increasingly more complex knowledge Domain, thus mimicking a *real-life* learning environment.

Our proposed model represented, to the best of our knowledge, a novel concept in the LA paradigm in which the complexity of the LA Environment is gradually increased so as to mimic teaching material with increasing difficulty. Incorporating stochastic information to model increasingly complex Domain knowledge, is also novel to the field of Intelligent Tutorial systems.

10.4.1 Future Work

With regard to future work, the application of this approach to model increasingly complex (stochastic) *real-life* Domain knowledge is open. In particular, the use of such a modeling paradigm for stochastic-type multiple choice questions that the Student answers is unsolved. Another distant future work is to port this approach into traditional Tutorial systems, but the modeling of the Domain knowledge for such systems will definitely not be trivial.

10.5 Modeling Teachers who *Provide* Hints

We believe that this Thesis enhances the basic LA paradigm, because we propose a methodology by which the convergence of the learning entity (i.e., the Student) can be influenced not only by the response that it received from the Environment (i.e., the Teacher), but also by the hints provided by the latter. These hints, provided by modifying the action probability vector of the LA, served to enable the Students to learn with additional *a priori* information, and also to handle more complex Domain material. We believe that this concept is unreported in the LA literature. Its practical applicability is obvious.

10.5.1 Future Work

First of all, we advocate the applicability of this "hint" strategy to all LA-based application solutions. The issue of providing hints to ergodic LA remains unsolved. Finally, we believe that our approach of providing *hints* can also be ported for use in traditional Tutorial Systems. By allowing the Teacher to present *hints* to *real-life* Students, they can hopefully be expected to benefit and learn more complex material. However, the question of how the hints can be provided in a real application domain is still open.

10.6 Modeling the "Learning Process" of a Teacher

Our Thesis also presented a new concept by which the Teacher himself can "learn" and improve his "teaching skills" while using the system. The characteristic of our Teacher within the Tutorial-*like* system is unique: He makes use of the Student's learning model (which is independently "learnt" and inferred) and the learning progress of the Students. Thereafter, he customizes his teaching according to the skills and abilities of each Student.

The Thesis demonstrated the hypothesis that the Teacher (Environment) himself can be designed in a manner that adapts to the particular needs and skills of each Student. This is, clearly, an important factor for any useful Tutorial or Tutorial-*like* system.

10.6.1 Future Work

Improving the Teacher's "teaching" abilities can have vast potential in Tutorial systems. The degree of the Teacher's "adaptability" in a Tutorial system can influence the practicality of the system. We believe that the concept of a Teacher's adaptability can be of use if it is properly ported to "traditional" Tutorial systems.

10.7 Overall Tutorial-*like* system

The basic premise of our research was that the entire system which we have advocated is new to the fields of LA and Tutorial systems. While the overall design and architecture of the system was given in Chapter 3, Chapter 9 included the details of the implementation of the prototype of the overall Tutorial-*like* system. This Chapter described how all the models described in the body of the Thesis functioned together, and how a session of the Tutoring for a Classroom of Students could be simulated.

The system, as we have designed it, permits the user to simulate a single Student or a Classroom of Students. These Students can chose to work independently or in discussion groups. The system also imparts to the Teacher, all the characteristics described in the above two subsections.

Finally, the system permits single runs or ensemble of experiments, and a graphical display of the corresponding performance criteria. Although this is just a prototype, we are not aware of any comparable systems.

10.7.1 Future Work

We foresee at least two main fundamental avenues for future research when it concerns issues related to the overall Tutorial-*like* system. The first involves incorporating specific Domain knowledge. In this case, incorporating all of these concepts into *real-life* Tutorial systems will be a massive undertaking. We believe that the applicability of this will be domain dependent. Secondly, the application of this strategy to stochastic systems, in which every module is conceptually a Student, is a more realistic goal.

We are open to collaborating with researchers in AI, cybernetics, and systems theory to achieve these goals.

Bibliography

[1] M. Agache. Families of estimator-based stochastic learning algorithms. Master's thesis, School of Computer Science, Carleton University, Ottawa, Canada, 2000.

[2] M. Agache and B. J. Oommen. Continuous and discretized generalized pursuit learning schemes. In S. Andraddttir, K. J. Healy, D. H. Withers, and B. L. Nelson, editors, *Proceedings of SCI-2000, the 4th World Multiconference on Systemics, Cybernetics*, pages VII:270–275, Orlando, FL, July 2000.

[3] M. Agache and B. J. Oommen. 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.

[4] S. R. Alpert, M. K. Singley, and P. G. Fairweather. Deploying intelligent tutors on the web: An architecture and an example. *International Journal of AI in Education*, 10:183–197, 1999.

[5] J. R. Anderson. The expert module. In M. C. Polson and J. J. Richardson, editors, *Foundations of Intelligent Tutoring Systems*, pages 21–53. Lawrence Erlbaum Associates Publishers, Hillsdale, New Jersey, 1988.

[6] A. Ansari and G. P. Papavassilopoulos. A generalized learning algorithm for an automaton operating in a multiteacher environment. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 29(5):592–600, October 1999.

[7] R. L. Atkinson, R. C. Atkinson, E. E. Smith, and D. J. Bem. *Introduction to Psychology.* Harcourt Brace Jovanovich, Fort Worth, TX, eleventh edition, 1993.

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

[9] N. Baba. *New Topics in Learning Automata Theory and Applications.* Spring-Verlag Berlin, Heidelberg, 1985.

[10] N. Baba and Y. Mogami. A new learning algorithm for the hierarchical structure learning automata operating in the nonstationary s-model random environment. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 32(6):750–758, December 2002.

[11] N. Baba and Y. Mogami. A consideration on the learning behaviors of the hierarchical structure learning automata operating in the nonstationary multiteacher environment – a basic research to realize an effective utilization of artificial neural networks in the nonstationary environment. In *Proceedings of the 34th Annual Conference of ISAGA*, pages 1021–1030, Chiba, 2003.

[12] N. Baba and Y. Mogami. Learning behaviors of the hierarchical structure stochastic automata operating in the nonstationary multiteacher environment. In *Knowledge-Based Intelligent Information and Engineering Systems: 9th International Conference, KES 2005*, pages 624–634, Melbourne, Australia, September 2005.

[13] A. Baena, M. Belmonte, and L. Mandow. An intelligent tutor for a web-based chess course. In P. Brusilovsky, O. Stock, and C. Strapparava, editors, *Proceedings of the Adaptive Hypermedia and Adaptive Web-Based Systems: International Conference, AH 2000*, pages 17–26, Trento, Italy, August 2000.

[14] P. Baffes and R. Mooney. Refinement-based student modeling and automated bug library construction. *Journal of AI in Education*, 7(1):75–116, 1996.

[15] A. G. Barto and C. W. Anderson. Cooperativity in networks of pattern recognizing stochastic learning automata. In K. S. Narendra, editor, *Adaptive and Learning Systems, Theory and Applications*. Plenum Press, New York, 1986.

[16] A. L. Baylor. Permutations of control: Cognitive considerations for agent-based learning environments. *Journal of Interactive Learning Research*, 12(4):403–425, 2001.

[17] J. Beck. Learning to teach with a reinforcement learning agent. In *Proceedings of The Fifteenth National Conference on AI/IAAI*, page 1185, Madison, WI, 1998.

[18] J. Beck, M. Stern, and E. Haugsjaa. Applications of AI in education. *ACM Crossroads*, 3(1), 1996.

[19] J. Beck, B. Woolf, and C. Beal. ADVISOR: A machine learning architecture for intelligent tutor construction. In *Proceedings of The Seventeenth National Conference on AI*, pages 552–557, 2000.

[20] S. Bhagat, L. Bhagat, J. Kavalan, and M. Sasikumar. Acharya: An intelligent tutoring environment for learning SQL. In *Proceedings of Vidyakash 2002-International Conference on Online Learning*, National Resource Center For Online Learning, NCST Mumbai, 2002.

[21] J. S. Brown, R. R. Burton, and J. De Kleer. Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II, and III. In D. Sleeman and J. S. Brown, editors, *Intelligent Tutoring Systems*. Academic Press, London, 1982.

[22] P. Brusilovsky and C. Peylo. Adaptive and intelligent web-based educational systems. *International Journal of AI in Education*, 13:156–169, 2003.

[23] H. L. Burns and C. G. Capps. Foundations of intelligent tutoring systems: An introduction. In M. C. Polson and J. J. Richardson, editors, *Foundations of Intelligent Tutoring Systems.* Lawrence Erlbaum Assoc., Hillsdale, New Jersey, 1988.

[24] R. R. Burton. Diagnosing bugs in a simple procedural skill. In D. Sleeman and J. S. Brown, editors, *Intelligent Tutoring Systems.* Academic Press, London, 1982.

[25] R. R. Burton and J. S. Brown. An investigation of computer coaching for informal learning activities. In D. Sleeman and J. S. Brown, editors, *Intelligent Tutoring Systems.* Academic Press, London, 1982.

[26] R. R. Burton and J. S. Brown. A tutoring and student modeling paradigm for gaming environments. *ACM SIGCSE Bulletin*, 8(1):236–246, 1982.

[27] J. R. Carbonell. AI in CAI: An artificial intelligence approach to computer-assisted instruction. *IEEE Transactions on Man-Machine Systems*, 11(4):190–202, 1970.

[28] B. Chandrasekharan and D. W. C. Shen. Stochastic automata games. *IEEE Transactions on System Science and Cybernetics*, SSC-5:145–149, 1969.

[29] B. A. Cheikes. Should ITS designer be looking for a few good agents? In N. Major, T. Murray, and C. Bloom, editors, *AI-ED Workshop on Authoring Shells for Intelligent Tutoring Systems.* AACE, 1995.

[30] H. Chen and M. Chau. Web mining: Machine learning for web applications. *Annual Review of Information Science and Technology (ARIST)*, 38:289–329, 2004.

[31] S. Cheng and H. Chang. Multicriteria automatic essay assessor generation by using TOPSIS model and genetic algorithm. In M. Ikeda, K. Ashley, and T. Chan, editors, *Intelligent Tutoring Systems: 8th International Conference, ITS 2006*, pages 11–20, Jhongli, Taiwan, June 2006.

[32] W. J. Clancey. Tutoring rules for guiding a case method dialogue. In D. Sleeman and J. S. Brown, editors, *Intelligent Tutoring Systems.* Academic Press, London, 1982.

[33] W. J. Clancey. *Knowledge Based Tutoring: The GUIDON Program.* MIT Press, Cambridge, MA, 1987.

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

[35] G. Cumming and J. Self. Collaborative intelligent educational system. In D. Bierman, J. Breuker, and J. Sandberg, editors, *Proceedings of the 4th International Conference on Artificial Intelligent and Education,* pages 73–80, Amsterdam, Netherlands, 1989.

[36] R. F. Drenick and R. A. Shahbender. Adaptive servomechanisms. *AIEE Transactions,* 76:286–292, 1957.

[37] A. A. Economides and A. Kehagias. The STAR automaton: Expediency and optimality properties. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics,* 32(6):723–737, December 2002.

[38] V. W. Eveleigh. *Adaptive Control and Optimization Techniques.* McGraw-Hill Book Company, New York, 1967.

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

[40] C. Frasson, L. Martin, G. Gouardères, and E. Aïmeur. LANCA: A distance learning architecture based on networked cognitive agents. In *Intelligent Tutoring Systems: 4th International Conference, ITS '98: Lecture Notes in Computer Science,* pages 593–603. Springer-Verlag, Berlin, 1998.

[41] C. Frasson, T. Mengelle, E. Aïmeur, and G. Gouardères. An actor-based architecture for intelligent tutoring systems. In *Intelligent Tutoring Systems: 3rd*

International Conference, ITS '96: Lecture Notes in Computer Science, pages 57–65. Springer-Verlag, Berlin, 1996.

[42] K. S. Fu and G. J. McMurtry. A study of stochastic automata as models of adaptive and learning controllers. Technical Report TR-EE 65-8, Purdue University, Lafayette, IN, 1965.

[43] I. Goldstein. Overlays: A theory of modeling for computer aided instruction. AI Memo 406, MIT, Cambridge, MA, 1977.

[44] I. Goldstein and S. Papert. Artificial intelligence, language and the study of knowledge. *Cognitive Science*, 1(1), 1977.

[45] H. A. Güvenir. An object-oriented tutoring system for teaching sets. *ACM SIGCSE Bulletin*, 27(3), 1995.

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

[47] A. A. Hashim, S. Amir, and P. Mars. Application of learning automata to image data compression. In K. S. Narendra, editor, *Adaptive and Learning Systems, Theory and Applications*. Plenum Press, New York, 1986.

[48] I. Hatzilygeroudis and J. Prentzas. Knowledge representation requirements for intelligent tutoring systems. In J. C. Lester, R. M. Vicari, and F. Paraguaçu, editors, *Intelligent Tutoring Systems: 7th International Conference, ITS 2004*, pages 87–97, Alagoas, Maceió, Brazil, 2004.

[49] S. Haykin. *Neural Networks: A Comprehensive Foundation*. IEEE Press/ Macmillan College Publishing Company, New York, 1994.

[50] N. Heffernan, E. Croteau, and K. Koedinger. Why are algebra word problems difficult? using tutorial log files and the power law of learning to select the

best fitting cognitive model. In J. C. Lester, R. M. Vicari, and F. Paraguaçu, editors, *Intelligent Tutoring Systems: 7th International Conference, ITS 2004*, pages 240–250, Alagoas, Maceió, Brazil, 2004.

[51] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning and Discovery.* MIT Press, Cambridge, MA, 1986.

[52] P. Holt, S. Dubs, M. Jones, and J. Greer. The state of student modeling. In J. Greer and G. McCalla, editors, *Student Modeling: The Key to Individualized Knowledge-Based Instruction.* Springer-Verlag, Berlin, 1994.

[53] U. Hoppe. Deductive error diagnosis and inductive error generalization for intelligent tutoring systems. *Journal of AI in Education*, 5(1), 1994.

[54] S. Ikebou, F. Qian, and H. Hirata. A multi-teacher learning automata computing model for graph partitioning problems. *The Transactions of The Institute of Electrical Engineers of Japan*, 123-C(4):832–838, April 2002.

[55] A. Jameson. Numerical uncertainty management in user and student modeling: An overview of systems and issues. *User Modeling and User Adapted Interaction*, 5(3-4), 1996.

[56] R. Kass. Student modeling in intelligent tutoring systems - implications for user modeling. In A. Kobsa and W. Wahlster, editors, *User Models in Dialog Systems.* Springer-Verlag, 1989.

[57] G. Kearsley. *AI and Instruction.* Addison-Wesley Publishing Company, MA, 1987.

[58] D. E. Koditscheck and K. S. Narendra. Fixed structure automata in multi-teacher environment. *IEEE Transactions on Systems, Man, and Cybernetics*, 7, 1977.

[59] K. R. Koedinger, J. R. Anderson, W. H. Hadley, and M. A. Mark. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8:30–43, 1997.

[60] T. Kohonen. Analysis of simple self organizing process. *Biological Cybernetics*, 44, 1982.

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

[62] S. Lakshmivarahan and M. A. L. Thathachar. Absolutely expedient learning algorithms for stochastic automata. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3:281–286, 1973.

[63] J. K. Lanctôt. Discrete estimator algorithms: A mathematical model of computer learning. Master's thesis, Department of Mathematics and Statistics, Carleton University, Ottawa, Canada, 1989.

[64] J. K. Lanctôt and B. J. Oommen. Discretized estimator learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6), November/December 1992.

[65] P. Langley and S. Ohlsson. Automated cognitive modeling. In *Proceedings of the Fourth National Conference of the American Association for Artificial Intelligence*, pages 193–197, 1984.

[66] R. S. Legaspi and R. C. Sison. Modeling the tutor using reinforcement learning. In *Proceedings of the Philippine Computer Science Congress (PCSC)*, pages 194–196, 2000.

[67] R. Lelouche. A collection of pedagogical agents for intelligent educational systems. In *Intelligent Tutoring Systems: 6th International Conference, ITS 2000: Lecture Notes in Computer Science*, pages 143–152. Springer-Verlag, Berlin, 2000.

[68] N. Makatchev, P. Jordan, and K. VanLehn. Modeling student's reasoning about qualitative physics: Heuristics for abductive proof search. In J. C. Lester, R. M. Vicari, and F. Paraguaçu, editors, *Intelligent Tutoring Systems: 7th International Conference, ITS 2004*, pages 699–709, Alagoas, Maceió, Brazil, 2004.

[69] C. Marsh and T. J. Gordon. The application of learning automata to controller design in slow-active automobile suspensions. *International Journal for Vehicle Mechanics and Mobility*, 24(8):597–616, 1995.

[70] A. Martens. Centralize the tutoring process in intelligent tutoring systems. In *Proceedings of the 5th International Conference on New Educational Environments, ICNEE*, Lucerne, Switzerland, May 2003.

[71] K. N. Martin and I. Arroyo. AgentX: Using reinforcement learning to improve the effectiveness of intelligent tutoring systems. In J. C. Lester, R. M. Vicari, and F. Paraguaçu, editors, *Intelligent Tutoring Systems: 7th International Conference, ITS 2004*, pages 564–572, Alagoas, Maceió, Brazil, 2004.

[72] L. G. Mason and X. Gu. Learning automata models for adaptive flow control in packet-switching networks. In K. S. Narendra, editor, *Adaptive and Learning Systems, Theory and Applications*. Plenum Press, New York, 1986.

[73] S. A. Mathan and K. R. Koedinger. An empirical assessment of comprehension fostering features in an intelligent tutoring system. In S. A. Cerri, G. Gouardères, and F. Paraguaçu, editors, *Intelligent Tutoring Systems: 6th International Conference, ITS 2002: Lecture Notes in Computer Science*, pages 330–343. Springer-Verlag, Berlin, 2002.

[74] G. I. McCalla. The central importance of student modeling to intelligent tutoring. In E. Costa, editor, *New Directions for Intelligent Tutoring Systems*. Spring-Verlag, Berlin, 1992.

[75] S. Misra. *Network Applications of Learning Automata*. PhD thesis, School of Computer Science, Carleton University, Ottawa, Canada, 2006.

[76] B. T. Mitchell and D. I. Kountanis. Reorganization scheme for a hierarchical system of learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14(2):328–334, 1984.

[77] A. Mitrovic, M. Mayo, P. Suraweera, and B. Martin. Constraint-based tutors: A success story. In L. Monostori, J. Váncza, and M. Ali, editors, *Engineering of Intelligent Systems: 14th International Conference on Industrial and Engineering Applications of AI and Expert Systems, IEA/AIE 2001*, pages 931–940, Budapest, Hungary, June 2001.

[78] S. Mukhopadhyay and M. A. L. Thathachar. Associative learning of boolean functions. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1008–1015, 1989.

[79] K. Najim and A. S. Poznyak. *Learning Automata Theory and Practice.* Elsevier Science Ltd., UK, 1994.

[80] K. S. Narendra and A. M. Annaswamy. *Stable Adaptive Systems.* Prentice-Hall, New Jersey, 1989.

[81] K. S. Narendra and K. Parthasarathy. Learning automata approach to hierarchical multiobjective analysis. Technical Report 8811, Electrical Engineering, Yale University, New Haven, Connecticut, 1988.

[82] K. S. Narendra and M. A. L. Thathachar. Learning automata: A survey. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14:323–334, 1974.

[83] K. S. Narendra and M. A. L. Thathachar. On the behavior of a learning automata in a changing environment with routing applications. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-10(5):262–269, 1980.

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

[85] K. S. Narendra and R. M. Wheeler. Recent development in learning automata. In K. S. Narendra, editor, *Adaptive and Learning Systems, Theory and Applications*. Plenum Press, New York, 1986.

[86] K. Naruse and Y. Kakazu. Strategy acquisition of path planning of redundant manipulator using learning automata. In *IEEE International Workshop Neuro-Fuzzy Control*, pages 154–159, 1993.

[87] M. G. Negoita and D. Pritchard. Using a virtual student model for testing intelligent tutoring systems. *Interactive Technology & Smart Education*, 1(3), 2004.

[88] H. S. Nwana. Intelligent tutoring systems: An overview. *AI Review*, 4:251–277, 1990.

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

[90] S. Ohlsson. Impact of cognitive theory on the practice of courseware authoring. *Journal of Computer Assisted Learning*, 9(4), 1993.

[91] S. Ohlsson and P. Langley. Psychological evaluation of path hypotheses in cognitive diagnosis. In H. Mandl and A. Lesgold, editors, *Learning Issues for Intelligent Tutoring Systems*. Springer, New York, 1988.

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

[93] B. J. Oommen. Absorbing and ergodic discretized two-action learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(2):282–293, March/April 1986.

[94] B. J. Oommen. Object partitioning using a hierarchy of stochastic automata. In *Proceedings of the 1990 IEEE International Conference on Systems, Man and Cybernetics*, pages 184–187, Los Angeles, CA, November 1990.

[95] B. J. Oommen and J. P. R. Christensen. ε-optimal discretized reward-penalty learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-18(3), May/June 1988.

[96] B. J. Oommen and E. V. de St. Croix. Graph partitioning using learning automata. Technical Report TR-250, School of Computer Science, Carleton University, Ottawa, Canada, July 1994.

[97] B. J. Oommen and E. V. de St. Croix. String taxonomy using learning automata. Technical Report TR-234, School of Computer Science, Carleton University, Ottawa, Canada, March 1994.

[98] B. J. Oommen and E. Hansen. The asymptotic optimality of discretized linear reward-inaction learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14(3), May/June 1986.

[99] B. J. Oommen and E. R. Hansen. The asymptotic optimality of discretized linear reward-inaction learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14:542–545, May/June 1984.

[100] B. J. Oommen and M. K. Hashem. On simulating tutorial-*like* systems using a learning automata philosophy. In *SummerSim'06, the 2006 Summer Simulation Multiconference*, Calgary, Canada, July/August 2006.

[101] B. J. Oommen and J. K. Lanctôt. Discretized pursuit learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(4), July/August 1990.

[102] B. J. Oommen and D. C. Y. Ma. Stochastic automata solutions to the object partitioning problem. *The Computer Journal*, 35:A105–A120, 1992.

[103] B. J. Oommen, G. Raghunath, and B. Kuipers. On how to learn from a stochastic teacher or a stochastic compulsive liar of unknown identity. In T. D. Gedeon

and L. C. C. Fung, editors, *AI 2003: Advances in AI: 16th Australian Conference on AI: Lecture Notes in Computer Science*, pages 24–40. Springer-Verlag, Berlin, 2003.

[104] G. I. Papadimitriou. Hierarchical discretized pursuit nonlinear learning automata with rapid convergence and high accuracy. *IEEE Transactions on Knowledge Data Engineering*, 6(4):654–659, August 1994.

[105] G. I. Papadimitriou, M. Sklira, and A. S. Pomportsis. A new class of ε-optimal learning automata. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 34(1):246–254, February 2004.

[106] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann Publishers, San Mateo, California, 1988.

[107] O. Popescu and K. R. Koedinger. Towards understanding geometry explanations. building dialogue systems for tutorial applications. *The 2000 AAAI Fall Symposium*, pages 80–86, 2000.

[108] J. Prentzas, I. Hatzilygeroudis, and J. Garofalakis. A web-based intelligent tutoring system using hybrid rules as its representational basis. In S. A. Cerri, G. Gouardères, and F. Paraguaçu, editors, *Intelligent Tutoring Systems: 6th International Conference, ITS 2002: Lecture Notes in Computer Science*, pages 119–128. Springer-Verlag, Berlin, 2002.

[109] K. Rajaraman and P. S. Sastry. Stochastic optimization over continuous and discrete variables with applications to concept learning under noise. *IEEE Transactions on Systems Man and Cybernetics Part A Systems and Humans*, 29(6):542–553, 1999.

[110] B. J. Reiser, J. R. Anderson, and R. G. Farrell. Dynamic student modelling in an intelligent tutor for LISP programming. In *Proceedings of IJCAI-85*, pages 8–14, Los Angeles, CA, August 1985.

[111] V. Reuer. Language processing and intelligent computer-assisted language learning. MiLCA-project, Institute of Cognitive Science, University of Osnabrück, Germany, 2004.

[112] J. Rickel. Intelligent computer-aided instruction: A survey organized around system components. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(1), 1989.

[113] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323, 1986.

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

[115] C. T. Santos, R. Frozza, A. Dhamer, and L. P. Gaspary. DÓRIS – pedagogical agent in intelligent tutoring systems. In S. A. Cerri, G. Gouardères, and F. Paraguaçu, editors, *Intelligent Tutoring Systems: 6th International Conference, ITS 2002: Lecture Notes in Computer Science*, pages 91–104. Springer-Verlag, Berlin, 2002.

[116] P. S. Sastry. *Systems of Learning Automata: Estimator Algorithms Applications.* PhD thesis, Dept of Electrical Engineering, Indian Institute of Science, Bangalore, India, June 1985.

[117] P. S. Sastry, V. V. Phansalkar, and M. A. L. Thathachar. Decentralized learning of nash equilibria in multi-person stochastic games with incomplete information. *IEEE Trans Systems, Man, and Cybernetics*, 24:769–777, May 1994.

[118] R. Schulmeister. *Hypermedia Learning Systems, Theory - Didactics - Design.* Oldenbourg, München, second edition, 1997.

[119] J. Self. Knowledge, belief and user modeling. In T. O'Shea and V. Sgurev, editors, *AI III: Methodology, Systems, Applications - Proceedings of the Third International Conference on AI: Methodology, Systems, Applications, (AIMSA '88)*, pages 3–9, Varna, Bulgaria, 1988.

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

[121] R. A. Silveira and R. M. Vicari. Developing distributed intelligent learning environment with JADE - Java agents for distance education framework. In S. A. Cerri, G. Gouardères, and F. Paraguaçu, editors, *Intelligent Tutoring Systems: 6th International Conference, ITS 2002: Lecture Notes in Computer Science*, pages 105–118. Springer-Verlag, Berlin, 2002.

[122] R. Sison, M. Numao, and M. Shimura. Using data and theory in multistrategy (mis)concept(ion) discovery. In *Proceedings of the Fifteenth International Joint Conference on AI*, pages 274–279, 1997.

[123] R. Sison and M. Shimura. The application of machine learning to student modeling: Survey and analysis. Technical Report TR96-0010, Dept. of Computer Science, Tokyo Institute of Technology, 1996.

[124] R. Sison and M. Shimura. Student modeling and machine learning. *International Journal of AI in Education*, 9(1-2), 1998.

[125] D. Sleeman. An attempt to understand students' understanding of basic algebra. *Cognitive Science*, 8:367–412, 1984.

[126] D. Sleeman and J. S. Brown. *Intelligent Tutoring Systems.* Academic Press, London, 1982.

[127] R. Stathacopoulou, M. Grigoriadou, G. D. Magoulas, and D. Mitropoulos. A neuro-fuzzy approach in student modeling. In P. Brusilovsky, A. Corbett, and F. de Rosis, editors, *9th International Conference, User Modeling 2003*, pages 337–341. Springer-Verlag, Berlin, 2003.

[128] A. Stevens, A. Collins, and S. Goldin. Misconceptions in students' understanding. In D. Sleeman and J. S. Brown, editors, *Intelligent Tutoring Systems.* Academic Press, London, 1982.

[129] P. Suppes. Some theoretical models for mathematics learning. *Journal of Research and Development in Education*, 1:5–22, 1967.

[130] P. Suraweera and A. Mitrovic. KERMIT: A constraint-based tutor for database modeling. In S. A. Cerri, G. Gouardères, and F. Paraguaçu, editors, *Intelligent Tutoring Systems: 6th International Conference, ITS 2002: Lecture Notes in Computer Science*, pages 377–387. Springer-Verlag, Berlin, 2002.

[131] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[132] M. A. L. Thathachar and M. T. Arvind. Parallel algorithms for modules of learning automata. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 28(1):24–33, February 1998.

[133] M. A. L. Thathachar and R. Bhakthavathsalam. Learning automata operating in parallel environment. *Journal of Cybernetics and Information Science*, 1, 1977.

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

[135] M. A. L. Thathachar and K. R. Ramakrishnan. A hierarchical system of learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(3):236–241, 1981.

[136] M. A. L. Thathachar and P. S. Sastry. A class of rapidly converging algorithms for learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15:168–175, 1985.

[137] M. A. L. Thathachar and P. S. Sastry. A new approach to designing reinforcement schemes for learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15, 1985.

[138] M. A. L. Thathachar and P. S. Sastry. Estimator algorithms for learning automata. In *Proceedings of the Platinum Jubilee Conference on Systems and Signal Processing*, pages 29–32, Dept. Electrical Engineering, Indian Institute of Science, Bangalore, India, December 1986.

[139] M. A. L. Thathachar and P. S. Sastry. Learning optimal discriminant functions through a cooperative game of automata. *IEEE Transactions on Systems, Man and Cybernetics*, 17(1):73–85, 1987.

[140] M. A. L. Thathachar and P. S. Sastry. Learning automata in stochastic games with incomplete information. In *System and Signal Processing*, pages 137–140. 1991.

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

[142] M. A. L. Thathachar and P. S. Sastry. *Networks of Learning Automata: Techniques for Online Stochastic Optimization.* Kluwer Acadmic Publishers, Boston, MA, 2004.

[143] S. G. Towns, P. J. FitzGerald, and J. C. Lester. Visual emotive communication in lifelike pedagogical agent. In B. P. Goettl, H. M. Halff, C. L. Redfield, and V. J. Shute, editors, *Intelligent Tutoring Systems: 4th International Conference, ITS '98: Lecture Notes in Computer Science*, pages 474–483. Springer-Verlag, Berlin, 1998.

[144] M. L. Tsetlin. On the behavior of finite automata in random media. *Automation and Remote Control*, 22:1210–1219, 1962.

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

[146] L. Uhr. Teaching machine programs that generate problems as a function of interaction with students. In *Proceedings of 1969 24th National Conference*, pages 125–134, 1969.

[147] C. Ünsal. *Intelligent Navigation of Autonomous Vehicles in an Automated Highway System: Learning Methods and Interacting Vehicles Approach.* PhD thesis, Virginia Polytechnic Institute and State University, VA, 1997.

[148] C. Ünsal, P. Kachroo, and J. S. Bay. Multiple stochastic learning automata for vehicle path control in an automated highway system. *IEEE Transactions on Systems, Man, and Cybernetics*, 29(1):120–128, January 1999.

[149] K. VanLehn, C. Lynch, K. Schulze, J. A. Shapiro, R. Shelby, L. Taylor, D. Treacy, A. Weinstein, and M. Wintersgill. The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education*, 15(3):147–204, 2005.

[150] V. I. Varshavskii and I. P. Vorontsova. On the behavior of stochastic automata with a variable structure. *Automation and Remote Control*, 24:327–333, 1963.

[151] M. F. Verdejo. Building a student model for an intelligent tutoring system. In J. Greer and G. McCalla, editors, *Student Modeling: The Key to Individualized Knowledge-Based Instruction*, pages 147–163. Springer-Verlag, Berlin, 1994.

[152] R. Viswanathan and K. S. Narendra. Games of stochastic automata. *IEEE Transactions on Systems, Man and Cybernetics*, 4:131–135, 1974.

[153] K. Warendorf and C. Tan. ADIS - an animated data structure intelligent tutoring system or putting an interactive tutor on the WWW. In *Proceedings of the 8th World Conference of the AIED Society*, Kobe, Japan, August 1997.

[154] E. Wenger. *AI and Tutoring Systems - Computational and Cognitive Approaches to the Communication of Knowledge.* Morgan Kaufmann Publishers, Los Altos, CA, 1987.

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

[156] M. Winter. Developing a group model for student software engineering teams. Master's thesis, University of Saskatchewan, Saskatoon, Saskatchewan, 2004.

[157] P. Woods and J. R. Hartley. Some learning models for arithmetic tasks and their use in computer-based learning. *British Journal of Educational Psychology*, 41:35–48, 1971.

[158] A. Yang, Kinshuk, and A. Patel. A plug-able web-based intelligent tutoring system. In S. Wrycza, editor, *Proceedings of the 10th European Conference on Information Systems*, pages 1422–1429, Wydawnictwo Uniwersytetu Gdanskiego, Gdañsk, Poland, June 2002.

[159] L. A. Zadeh. On the definition of adaptivity. *Proceedings of the IEEE*, 51:469–470, March 1963.

[160] C. Zinn, J. D. Moore, M. G. Core, S. Varges, and K. Porayska-Pomsta. The BE&E tutorial learning environment (BEETLE). In *Proceedings of the 7th Workshop on the Semantics and Pragmatics of Dialogue (DiaBruck)*, pages 209–210, Saarbrücken, Germany, 2003.