# Using POMDP-based Reinforcement Learning for Online Optimization of Teaching Strategies in an Intelligent Tutoring System

by

Pengfei Zhang

A thesis

presented to

The University of Guelph

In partial fulfilment of requirements

for the degree of

Master of Science

in

Computer Science

Guelph, Ontario, Canada

# ABSTRACT

# USING POMDP-BASED REINFORCEMENT LEARNING FOR ONLINE OPTIMIZATION OF TEACHING STRATEGIES IN AN INTELLIGENT TUTORING SYSTEM

**Pengfei Zhang**

**University of Guelph, 2013**

**Advisor:**

**Professor Fangju Wang**

This thesis is an investigation of "Using POMDP-based Reinforcement Learning for Online Optimization of Teaching Strategies in an Intelligent Tutoring System". A challenge in building an intelligent tutoring system (ITS) is to create and maintain an optimal teaching strategy. We cast an ITS as a partially observable Markov decision process (POMDP), and apply a reinforcement learning (RL) algorithm to learn the optimal teaching strategy through interactions between the system and the students. The optimal teaching strategy is chosen correctly and efficiently in tutoring a student, it is also learned and maintained in an online model. We present an RL algorithm based on POMDP for learning optimal teaching strategy, then describe the experiments and analyse the experimental results. The experiment has showed that the technique can remarkably improve an ITS's teaching performance.

## Acknowledgements

I wish to give thanks to a few of those who so generously gave their time and effort in helping me prepare my thesis. Without these people, this thesis would not be what it is.

First and foremost, I am particularly grateful to my advisor Dr. Fangju Wang for his support and guidance during my Master programme. Without his professional guide, I could not found my direction and finished this thesis.

As well, I would like to thank my Advisory Committee member, Dr. Fei Song for his advice and support.

I would also like to thank Graduate Coordinator, Dr. Gary Grewall for his support to my studies.

Meanwhile, I would like to thank my family who have helped and supported me during my studies.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 ITS and teaching strategy

An *intelligent tutoring system* (*ITS*) is a computer system that teaches its human student usually on a turn-by-turn basis. An ITS interprets student input and responds to the student based on its interpretations. ITS have started to and will broadly be used in computer-based tutoring and training. An ITS may work as a tutor or examiner, or both. When working as a tutor, it answers questions from the student. When working as an examiner, it asks questions and evaluates student answers.

ITSs have remarkable strengths, including flexibility and low costs. With an ITS, a student may study a subject that is not taught in a local school. It may be easy

for the student to incorporate the studies into his/her daily schedule, and to control the pace. Compared with human teachers, ITSs have major advantages. In many situations, studying with an ITS may be less expensive than attending a class. A well-designed ITS may provide more complete and up-to-date knowledge, and it is easier to update an ITS's knowledge base. An ITS may stand alone, it may also be combined with a system of speech, graphics, animation, etc. Besides tutoring school subjects, ITSs can also be used for product user training, trouble shooting, and so on.

A *teaching strategy* is important for an ITS that teaches complex subjects. In this research, we address the creation of a teaching strategy for an ITS that teaches a subject in which a concept may have many prerequisite concepts. To understand a concept, the student must first understand its direct and indirect prerequisites. To teach a student about a concept, an ITS may need to teach the prerequisites that the student does not understand. If the system chooses to teach too many prerequisites that the student already understands, the student may become impatient and the teaching may be inefficient. If the system does not teach the required prerequisites, the student may become frustrated and the teaching may hardly be understandable. In this research, a teaching strategy is one that guides the system to choose the starting points (concepts) in answering questions, based on the study states of the student.

A *study state* is an abstract representation of what the student already understands and what the student does not understand, in studying a subject. When the system knows exactly the study state of the student, it should not have a major difficulty to choose a good answer to a question. However, in many practical situations, a

system does not always know exactly the student's study state. To develop an ITS for practical applications, we need a technique that enables the system to make a right decision (to choose a right answer) when it is uncertain about the student's study state. This is the problem we try to solve in our reearch.

## 1.2 Research objective and approach

In the thesis research, we develop a technique that equips an ITS with a teaching strategy that enables the system to answer student questions in the most suitable ways, even when the system does not have the exact information about the student's study states. The technique also enables the system to continuously improve the teaching strategy when it applies the strategy to teach students.

Partially observable Markov decision process (POMDP) is a useful tool for making decisions when there is uncertainty in observing states, for example in observing student study states.

POMDP is a kind of Markov decision process (MDP). In an MDP, the information required for making a decision is available in the current state. To make a decision, an agent does not need information from the states other than the current one. In terms of state observability, MDPs can be classified into completely observable Markov decision process (COMDP), and partially observable Markov decision process (POMDP). COMDP is suitable for applications in which states are completely observable to the decision-making agent, while POMDP is suitable for applications in which states are not, or only partially, observable to the agent, and thus the agent is uncertain about

its current states.

We choose POMDP as the tool to develop our technique. In our research, we investigate the suitability of applying POMDP for building an ITS, and develop a reinforcement learning (RL) algorithm based on POMDP. The algorithm creates and improve teaching strategies during system user interactions.

## 1.3 Thesis Organization

In Chapter 2, we review reinforcement learning and POMDP research work that has been done in building ITSs.

Chapter 3 describes the technical background of reinforcement learning and POMDP in general.

In Chapter 4, we introduce the POMDP for building an ITS, which is the core design in this thesis.

Chapter 5 presents the implementation of the technique, experimental design, experiments, and result anslysis.

Chapter 6 summarizes the major contributions of this research, and lists the possible future research.

# Chapter 2

# Related Work

## 2.1 Reinforcement Learning in ITSs

Reinforcement learning (RL) has been applied to develop ITSs in particular. Litman and co-workers developed a spoken dialogue system called ITSPOKE using RL[21]. A Why2-Atlas tutoring system in text is used for the system.Typically, the system gives some qualitative physical questions, the student answers them with his natural language text,system with ITSPOKE will then get the student involved in a spoken dialogue model,to answer the questions, to correct the wrong ideas and explain further if it is needed.

In [4], a human to human corpus and a human to computer corpus are used to

examine the correlations with each other for dialogue characteristics.Both corpus are implanted into the tutoring system manually. With human to computer corpus, we can see that the student inputs can be displayed reasonably with computer feedback, they both are correlated during the tutoring study process.

In [12], RL algorithm is used in a education teaching pedagogical model so that the system can find out what best policy is that is suitable to the students. One of the major features of this method is that the system can improve its policy based on the feedbacks it already gotten previously from other students in similar situations and backgrounds.During the process, author studied three situations: accurate policies were applied for the learning; the RL to AIES was applied to application during exploration or exploitation process; AIES training phase was reduced by a special method.

Researchers also used RL to investigate student emotion in computer-based studying. In the work reported in [1], in order to find out the student's feedback to the machine during the tutoring process, system and user performance features were combined together into the learning process. The authors of [1] argued that student emotion detection was important in building a ITS. Information about student emotion could be used to determine the timing to pass the dialogue to the student and so on.

In [14], rather than developing hand-written decision rules, it adopts a data-driven approach by collecting user feedback on a variety of possible Temporal Expressions in terms of task success, ambiguity, and user preference.The data collected in this work is freely available to the research community. These data were then used to train a simulated user and a reinforcement learning policy that learns an adaptive Temporal

Expression generation strategy for a variety of contexts.

Through interacting with users [22], author designed a tutoring system in which the student and system can communicate with each other interactively.The system gives the feedback to the student tentatively,meanwhile, it also gives some descriptions for some key words in different topics. The student can ask some more questions if he is not happy with the tentative answers from the system based on those key works given by the system.A hierarchy of key words based on Markov decision process were annotated for the whole system design which is often used in spoken dialogue system for tutoring.With RL, all those key words are ranked in order to get higher successful tentative rates, meanwhile make the number of communications as less as possible.

An RL approach were developed in order to find out the import tutoring features for ITS.Through experiments reported in [27],author found that when the state space is defined in ITS, other tutoring factors are very important and have more impact to the whole process, such as the system actions, student emotion,repeated the questions and answers between student and system,system performance, etc.

In the work reviewed above, and in the other existing work for applying RL in building ITSs, RL was mainly used as an off-line solution for tutoring and tutoring policy definition.

## 2.2 POMDP in ITS

With PMODP, it is possible for us to predict user's action in ITS and plan even a state is uncertain.It is mainly based on the assumption based on the current student

state,gives a possible reaction and goes to the next physical state.

This technique is characterized by maintaining a distribution over many possible hypotheses of the current dialogue state.

In [15] two POMDP representations are used: state queues and observation chains. It use ITS task properties and let POMDPs scale to represent over 100 independent learner features. A real-world military training problem is given as one example. A human study ($n = 14$) provides initial validation for the model construction. Finally, evaluating the experimental representations with simulated students helps predict their impact on ITS performance. The compressed representations can model a wide range of simulated problems with instructional efficacy equal to lossless representations. With improved tractability, POMDP ITSs can accommodate more numerous or more detailed learner states and inputs.

The algorithm developed by Williams and Young [35] was a typical one in which POMDP was used for SDS.For each stage, the student state is uncertain but in a physical state, the state information is referred as *belief state* $b$, which is a rate of possibility in the special physical state.With a special time stamp,both the learning process and system solving process is in an uncertain state. The belief state is in a vector as $(b(s_1), b(s_2), ...)^T$, the vector space is separated into different regions. As ambiguity could happen during the tutoring process, the ambiguity issues are fixed through the those belief state vectors and gives the student a best answer as the system can.

In [10],based on COMDP and POMDP,a SmartWheeler dialogue manager is designed with the previous dialogue histories.The tutoring system will get those histories into its system, then get POMDP involved, it creates two different models, one

is based on the keywords and another one is based on the student's questions.The keywords and student questions are compared based on long term rewords through POMDP algorithm.

For PMODP, student state is unknew at each phase, we only have the belief state and observation to calculate the next step,see [34]).We can got the observation from supervised learning [29],or RL [17]. Main difference between the approaches is that supervised learning is optimized which supposes the tutoring is completed successfully while the RL approach is to predict the best answer to the student.

In the work reviewed above, POMDP is used off-line and the algorithms are used only for fixed strategy.

# Chapter 3

# Technical Background: RL and POMDP

## 3.1 Reinforcement Learning

### 3.1.1 Components and Functions

Reinforcement learning (RL) is a technique of machine learning suitable for applications in which the learning agent interacts with the environment. An RL algorithm can be represented as tuple $(S, A, T, \rho)$, where $S$ is a set of *states*, $A$ is a set of *actions*,

$T$: $S \times A \times S \rightarrow [0, 1]$ defines *state transition probabilities*, $\rho$: $S \times A \times S \rightarrow \Re$ is the *reward function* where $\Re$ is a set of *rewards*.

At time $t$, the learning agent is in state $s_t \in S$, where it takes action $a_t \in A$, which causes the agent to enter state $s_{t+1} \in S$. In the terminology of RL, taking action $a_t$ in state $s_t$ causes *state transition* from $s_t$ into $s_{t+1}$. The probability of state transition from $s_t$ into $s_{t+1}$ after $a_t$ is taken is $P(s_{t+1}|s_t, a_t)$. After a state transition, the agent receives reward $r_{t+1} \in \Re$. The reward received by the agent after it takes $a_t$ in $s_t$ and enters $s_{t+1}$ is calculated as

$$r_{t+1} = \rho(s_t, a_t, s_{t+1}). \tag{3.1}$$

Policy $\pi$ is another important component in an RL algorithm. A policy has two different forms: deterministic $\pi(s)$ and stochastic $\pi(s, a)$. The deterministic $\pi$ can be used to choose actions for the learning agent to take. When $\pi$ is optimal, for state $s_t$ it returns action $a_t$, that is,

$$a_t = \pi(s_t), \tag{3.2}$$

which maximizes the long term *return* $R_t$ defined as

$$R_t = \sum_{k=0}^{n} \gamma^k r_{t+k+1} \tag{3.3}$$

where $r_i \in \mathcal{P}$ is a reward defined in (3.1) ($i = t+1, t+2, ...$), and $\gamma$ is a future reward *discounting factor* ($0 \leq \gamma \leq 1$).

The deterministic $\pi(s)$ is calculated as

$$\pi(s) = \arg\max_a Q^\pi(s, a). \tag{3.4}$$

$Q^\pi(s, a)$ is a value function. It will be discussed later. The stochastic $\pi(s, a)$ is defined

as:

$$\pi(s, a_k) = \frac{Q^\pi(s, a_k)}{\sum_{i=1}^{n} Q^\pi(s, a_i)} \tag{3.5}$$

where $n$ is the number of possible actions that can be taken in state $s$, and $1 \leq k \leq n$. When $\pi$ is optimal, $\pi(s, a)$ returns the probability that $a$ may maximize the long term benefit when it is taken in $s$.

$Q^\pi(s, a)$ is the *action-value function* given $\pi$. It evaluates the expected return if action $a$ is taken in $s$ and the subsequent actions are chosen by $\pi$. $Q^\pi(s, a)$ can be defined and calculated as

$$Q^\pi(s, a) = E[R|s, \pi, a] = \sum_{s'} P(s'|s, a) V^\pi(s') \tag{3.6}$$

where $E[\ ]$ denotes calculating the expected value, $R$ is the return defined in (3.3), $s'$ is the state that the agent enters after it takes $a$ in $s$, $P(s'|s, a)$ is the probability of transition from $s$ to $s'$ after $a$ is taken, and $V^\pi(s)$ is the *state-value function* that evaluates the expected return of $s$ given policy $\pi$. $V^\pi(s)$ can be defined and calculated as

$$\begin{aligned} V^\pi(s) &= E[R|s, \pi] \\ &= E_\pi[\sum_{k=0}^{n} \gamma^k r_{t+k+1} | s_t = s] \\ &= \sum_a \pi(s, a) \sum_{s'} P(s'|s, a)[\mathcal{R}(s, a, s') + \gamma V^\pi(s')] \end{aligned} \tag{3.7}$$

where $E_\pi[\ ]$ denotes calculating the expected value given that the agent follows policy $\pi$ hereafter, $\gamma$ is a future reward discounting factor, and $\mathcal{R}(s, a, s')$ is the *expected reward* when transiting from $s$ to $s'$ after $a$ is taken.

### 3.1.2 Policy improvement

Policy improvement is to generate a better policy from the current policy. By updating action-value function $V^\pi$ and state-value function $Q^\pi$ , an RL policy $\pi$ can be improved.

Let $\pi$ be the current policy, and $\pi(s)$ be the action maximizing $V^\pi(s)$. In state $s$ if there is an action $a \neq \pi(s)$, we may expect that there exists a policy $\pi'$ such that $a = \pi'(s)$. If $\forall s \in S$,

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s), \tag{3.8}$$

then based on the policy improvement theorem [26], $\pi'$ must be as good as, or better than, $\pi$. That is, $\forall s \in S$

$$V^{\pi'}(s) \geq V^\pi(s). \tag{3.9}$$

The task of *policy improvement* is to find such a $\pi'$.

Policy $\pi'$ can be found by improving $\pi$ through the *policy iteration process* [26], given in the following procedure.

1. Initialization
   Initialize $V^\pi$ based on $\pi$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in S$
   $v \leftarrow V^\pi(s)$
   $a \leftarrow \pi(s)$
   $V^\pi(s) \leftarrow \sum_{s'} P(s'|s, a)[\mathcal{R}(s, a, s') + \gamma V^\pi(s')]$
   $\Delta \leftarrow \max(\Delta, |v - V^\pi(s)|)$
   until $\Delta < \theta$

3. Policy Improvement
   policy_stable $\leftarrow$ true
   For each $s \in S$

$b \leftarrow \pi(s)$
$\pi(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^{\pi}(s')]$
If $b \neq \pi(s)$ then policy_stable $\leftarrow$ false
If policy_stable, then stop; else go to 2

## 3.2 Reinforcement Learning based on POMDP

### 3.2.1 COMDP vs POMDP

Conventional RL is based on Markov decision process (MDP). In an MDP, states are completely observable. MDP is also called *completely observable Markov decision process* (*COMDP*).

In an RL algorithm based on COMDP, a learning agent is completely certain about the current state $s$ it is in, and is also certain about the new state $s'$ after it takes action $a$ in $s$. $a$ is a function of $s$, i.e. $\pi(s)$, and the transition to $s'$ is a function of $s$ and $a$. Also, reward $r$ is a function of $s$, $a$, and $s'$. Certainty about states is essential for applying the conventional RL. For applications in which states are not fully observable, a conventional RL algorithm may not be suitable.

POMDP (Partially Observable Markov Decision Process) provides a better basis for applying reinforcement learning to applications in which states are not observable, or only partially observable. In many applications, at a point of time, the agent does not know exactly what state it is in, and after it takes an action, it does not know what new state it is in either. POMDP can help the agent to make decisions with uncertainty in viewing the states.

### 3.2.2 Main components in POMDP

A POMDP can be represented as tuple $(S, A, T, \rho, O, Z, b_0)$.

- $S$ is a set of states.
- $A$ is a set of actions.

- $T : S \times A \times S \rightarrow [0.1]$ defines state transition probabilities. $P(s'|s, a)$ is the probability of transition from $s$ to $s'$ after $a$ is taken in $s$. For each pair of $s$ and $a$, $\sum_{s' \in S} P(s'|s, a) = 1$.

- $\rho : S \times A \times S \rightarrow \Re$ is the *reward function*, where $\Re$ is a set of rewards. $\rho(s, a, s')$ returns the reward after the agent takes $a$ in $s$ and enters $s'$.

- $O$ is a set of *observations*.

- $Z : A \times S \rightarrow O$ defines *observation probabilities*. $P(o|a, s')$ denotes the probability that the agent observes $o$ after taking $a$ and enters $s'$. For each pair of $a$ and $s'$, $\sum_{s' \in S} P(o|a, s') = 1$. Note $P(o|a, s')$ is a probability of $o$ given $a$ and $s'$, not just $s'$.

- $b_0$ is the *initial belief state*.

Of the components in the tuple, $S$, $A$, $T$, and $\rho$ are the same as the counterparts in the conventional RL, whereas $O$, $Z$ and $b_0$ are new. In POMDP, $s \in S$ is not observable or not completely observable to the learning agent. The agent can only observe $o \in O$. To the agent, information about states is represented by a distribution over states, referred to as *belief state* denoted by $b$. In the following, when it is necessary to distinguish $s \in S$ from $b$, we refer to $s$ as a *physical state*.

At a point of time, the agent is in a physical state (the current state), however it does not know which is the current state. The information about states available to the agent is $b$, which is a vector

$$b = [b(s^1), b(s^2), ..., b(s^N)] \tag{3.10}$$

where $s^i \in S$ $(1 \leq i \leq N)$ is the $i$th physical state, $N$ is the number of states in $S$, and $\sum_{i=1}^{N} b(s^i) = 1$. $b(s^i)$ is the probability that $s^i$ is the current state. (Note we use superscripts of $s$ to number physical states, and use subscripts of $s$ to indicate time steps. The only exception is we use $s'$ to indicate the state in the next time step.)

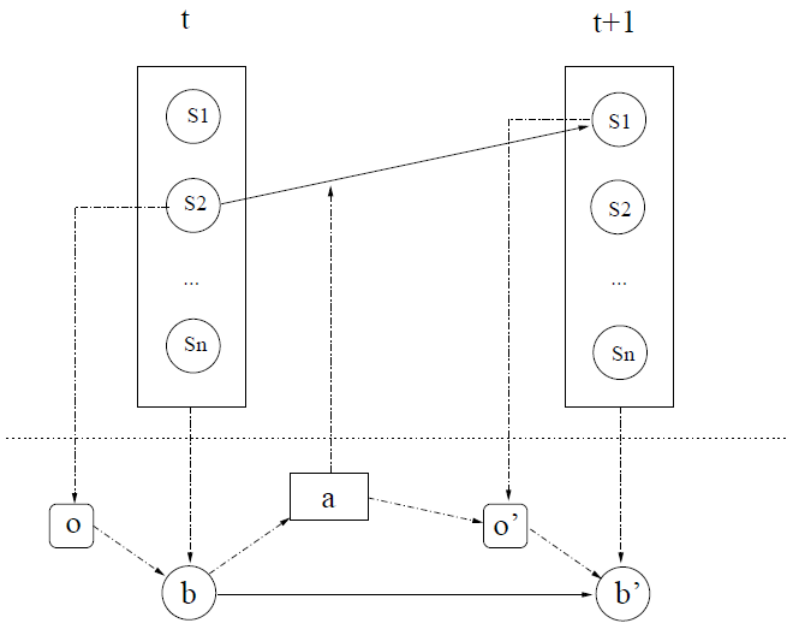


Figure 3.1: Physical state space, observations, belief states, action, and state transition.

Based on the state information in $b$, the agent's policy chooses an action $a$. After the agent takes $a$, the agent enters state $s'$, and observes $o$. The total probability of observing $o$ after taking $a$ is

$$P(o|a) = \sum_{s \in S} b(s) \sum_{s' \in S} P(s'|s,a) P(o|a,s'). \tag{3.11}$$

The expected reward after the agent takes $a$ is calculated as

$$\mathcal{R}(a) = \sum_{s \in S} b(s) \sum_{s' \in S} r(s,a,s'), \tag{3.12}$$

where $r(s,a,s') = \rho(s,a,s')$ is the reward that the agent receives after it takes $a$ in $s$ and enters $s'$.

After the agent takes $a$ in belief state $b$, the information of the new state will then be represented by belief state vector $b'$. In $b'$, an element is $b'(s')$, calculated as

$$b'(s') \quad = \quad \sum_{s \in S} b(s) P(s'|s,a) P(o|a,s') / P(o|a)$$

$$= \frac{P(o|a,s')\sum_{s\in S} b(s)P(s'|s,a)}{P(o|a)} \tag{3.13}$$

where $P(o|a)$ defined in (3.11) is used as a normalization constant so that the elements in $b'$ sum to one. Note that Eqn (3.11) can be rewritten as

$$P(o|a) = \sum_{s'\in S} P(o|a,s') \sum_{s\in S} b(s)P(s'|s,a). \tag{3.14}$$

For any $b$, the successor $b'$ depends only on $a$ and $o$. This fact can be expressed as

$$b'(s') = P(s'|a,s',o).$$

Each individual $b'(s') \in b'$ depends on $b$, $a$ and $o$, as in (3.13). That is, for each $s'$, $b'(s')$ is calculated based on the current belief state, the action taken, and the observation.

### 3.2.3 Solving a POMDP

Solving a POMDP is to find a policy $\pi$ such that the expected sum of discounted rewards is maximized.

A policy can be represented as a set of policy trees[31]. The task of solving a POMDP can be conducted by finding the optimal policy trees. Each policy tree is associated with a $V$ function. In the following, we denote the value function for executing policy tree $p$ as $V_p$.

For physical state $s$:

$$V_p(s) = \mathcal{R}(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \sum_{o\in O} P(o|a,s')V_{p(o)}(s') \tag{3.15}$$

where $a$ is the root action of policy tree $p$, $\mathcal{R}(s,a)$ is the expected immediate reward after $a$ is taken in $s$, $o$ is the observation after $a$ is taken, $p(o)$ is the subtree in $p$ which is connected to the root action by an edge labeled $o$, and $\gamma$ is a reward discounting factor. The second term in the expression on the right side is the discounted expected value of future states. $\mathcal{R}(s,a)$ is defined as

$$\mathcal{R}(s,a) = \sum_{s'\in S} P(s'|s,a)\mathcal{R}(s,a,s') \tag{3.16}$$

where $\mathcal{R}(s, a, s')$ is the expected immediate reward after the agent takes $a$ in $s$ and enters $s'$.

$V_p(s)$ can be understood by comparing Eqn (3.15) with Eqn (3.7). Eqn (3.7) can be rewritten as:

$$\begin{aligned} V^\pi(s) &= \sum_{a\in A} \pi(s,a) \sum_{s'\in S} P(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^\pi(s')] \\ &= \sum_{a\in A} \pi(s,a)[\sum_{s'\in S} P(s'|s,a)\mathcal{R}(s,a,s') + \gamma \sum_{s'\in S} P(s'|s,a)V^\pi(s')] \end{aligned} \quad (3.17)$$

In Eqn (3.17), the first term in the [ ] brackets represents the expected immediate reward for a given $s$ and a given $a$, and the second term is the discounted expected value of future states. The first term is the same as the right hand side of Eqn (3.16). Compare Eqns (3.15) and (3.17), we can see that the main difference is in the discounted expected value of future states. In Eqn (3.15), observations are taken into consideration.

The value of belief state $b$ given policy $p$, denoted by $V_p(b)$, is the value when the agent executes policy tree $p$ in $b$

$$V_p(b) = \sum_{s\in S} b(s)V_p(s). \quad (3.18)$$

$V_p(b)$ is an expectation over all the physical states given $p$. If we define $V_p$ as a vector with $V_p(s^i)$ $(i = 1, 2, ..., N)$ being its elements

$$V_p = [V_p(s^1), V_p(s^2), ..., V_p(s^n)], \quad (3.19)$$

then from Eqn (3.18), $V_p(b)$ can be represented as a dot product of $b$ and $V_p$:

$$V_p(b) = b \cdot V_p. \quad (3.20)$$

At different time steps, the learning agent may be in different physical states. From the perspective of the agent, it is in different belief state. For belief state $b$, there is an optimal policy tree $p$, which maximizes the value of the belief state:

$$V(b) = \max_{p\in\mathcal{P}} V_p(b) = \max_{p\in\mathcal{P}} b \cdot V_p \quad (3.21)$$

The policy $\pi(b)$ (approximated by a policy tree) is a function of $b$, returning a policy tree that maximize the value of $V(b)$:

$$\pi(b) = \hat{p} = \arg\max_{p \in \mathcal{P}} V_p(b) \tag{3.22}$$

# Chapter 4

# POMDP for Building an ITS

## 4.1 An Overview

In this chapter, we present our technique of POMDP for building an ITS. We cast an ITS as a partially observable Markov decision process (POMDP), and apply a reinforcement learning (RL) algorithm to learn the optimal teaching strategies through interactions between the system and the students. We define states based on the important concepts in the subject taught by the system. A *state* represents the agent's knowledge about a *study state* of the student. An *action* is an action taken by the agent, like answering a question. An *observation* is an action taken by the student and observed by the agent, like asking a question. An action by the agent causes a *state transition*.

The *teaching strategy* is modeled by the *policy* of the agent. To answer a student question, the agent applies the strategy to choose a suitable answer, based on the agent's knowledge about the student study state. When the study state is uncertain, the decision is based on the agent's *belief state*. The teaching strategy (modeled by the policy) is created as a function of the agent's belief state. The policy is first initialized by using training data, and updated continuously during the process of teaching.

## 4.2 States

In POMDP, at any point of time, there is a *current state*, which is a physical state. In our ITS architecture, it is the state the agent is in. The current state represents the agent's knowledge about the current study state of the student. Defining states is a core task in designing a POMDP-based RL algorithm. States and state transitions form the framework in which a learning agent conducts its jobs.

We define the states based on important concepts in the subject of tutoring. For example, in tutoring the basic level knowledge of software, the important concepts include "binary digit", "bit", "byte", "data", "computer file", "instruction", "programming language", "machine language", "assembly language", "high-level language", "query language", "program", "application program", "database", and so on.

Concepts are related to each other. To study a concept, a student must first understand some other concepts. We call the latter *prerequisites*. For example, to understand "computer file", a student has to understand "data" first. Thus "data" is a prerequisite of "computer file". A prerequisite concept may have its own prerequisites. For example, "data" has prerequisites of "bit" and "byte". A concept may have a number of direct and indirect prerequisites. Here, a direct prerequisite is one that appears in the definition or description of another concept. The definitions/descriptions of the sample concepts can be found in the appendix of the thesis.

The prerequisite relationships can be represented in a directed acyclic graph (DAG), in which the nodes are concepts and the edges are relationships of direct prerequisites. Figure 4.1 illustrates such a DAG for a subset of the concepts in tutoring software basic. We will see that information about concept prerequisites is useful in defining states and eliminating invalid states.

For each concept, we define two *conditions*. Let $C$ be the name of a concept. We have

- the *understand* condition, denoted by $\surd C$, indicating that the student under-

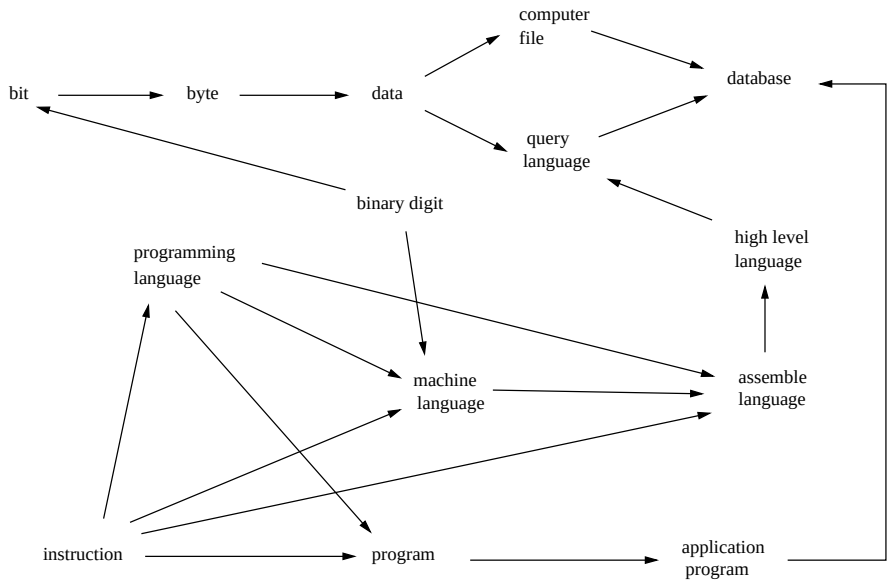


Figure 4.1: The directed acyclic graph (DAG) representing a subset of concepts in basic software tutoring and their prerequisite relationships.

stands the concept, and

- the *not understand* condition, denoted by $\neg C$, indicating that the student does not understand the concept.

The conditions are binary. Boolean conjunctive formulae of concept conditions can be constructed to represent study states of a student, with respect to the concepts. For instance, $(\surd C_1 \wedge \surd C_2 \wedge \neg C_3)$ can be used to represent the study state that the student understands concepts $C_1$ and $C_2$ but not concept $C_3$. In this thesis, we omit the $\wedge$ operator. We thus write the above formula as $(\surd C_1 \surd C_2 \neg C_3)$.

Now we come to the state definition in our ITS. In the architecture, a state is for a *study state* of the student, which specifies the concepts the student understands and the concepts the student does not understand. The representation of a state is a formula as presented above. Assume in a subject there are three concepts $C_1$, $C_2$, and $C_3$. $(\surd C_1 \surd C_2 \neg C_3)$ represents the study state that the student understands $C_1$ and $C_2$, while he does not understand $C_3$. $(\surd C_1 \surd C_2 \surd C_3)$ represents the study state

that the student understands all the three concepts. We call the formula associated with a state the *state expression* of the state.

When there are $N$ concepts in the subject, a state expression contains conditions of all the $N$ concepts, denoted by $(\mathcal{C}_1\mathcal{C}_2\mathcal{C}_3...\mathcal{C}_N)$, in which $\mathcal{C}_i$ $(1 \leq i \leq N)$ may take the values of $\surd C_i$ or $\neg C_i$. In state expressions, the concepts are sorted by applying a topological sorting algorithm to the DAG that represents the prerequisite relationships. In a topologically sorted state expression (a concept list), all the direct and indirect prerequisites of a concept are on the left hand side of it. That is, in state expression $(\mathcal{C}_1...\mathcal{C}_{i-1}\mathcal{C}_i...\mathcal{C}_N)$, all the direct and indirect prerequisites of $\mathcal{C}_i$ are in $\mathcal{C}_1$ through $\mathcal{C}_{i-1}$. As an example, the following is a result when the concepts in Figure 4.1 are topologically sorted:

$$\text{BD BI BY DA FI IN PL ML AL HL QL PR AP DB} \tag{4.1}$$

where BD stands for "binary digit", BI for "bit language", BY for "byte", DA for "data", FI for "file", IN for "instruction", PL for "programming language", ML for "machine language", AL for "assembly language", HL for "high-level language", QL for "query language", PR for "program", AP for "application program", and DB for "database".

When the number of concepts is $N$, we have $2^N$ possible state expressions, which is exponential. However, the actual number of states is much smaller than $2^N$. The reason is that most expressions are for invalid states. For example, $(\surd C_1 \neg C_2 \surd C_3...)$ is for an invalid state when $C_2$ is a prerequisite of $C_3$. Assume that $C_2$ is "bit" and $C_3$ is "byte". The expression represents an invalid state because a student could not correctly understand "byte" without knowledge of "bit", which is a prerequisite of "byte".

Assume we have a topologically sorted expression $(\mathcal{C}_1\mathcal{C}_2...\mathcal{C}_N)$, and for any $\mathcal{C}_i$ $(1 < i \leq N)$, all the concepts in $\mathcal{C}_1$ through $\mathcal{C}_{i-1}$ are its direct or indirect prerequisites. The expression is for a valid state if and only if when $\mathcal{C}_k$ is $\surd C_k$ $(1 < k \leq N)$, any $\mathcal{C}_i$ is $\surd C_i$ $(1 \leq i < k)$. It can be calculated that the number of valid expressions is $N$ in

this case. In many other cases, some concepts in $\mathcal{C}_1$ through $\mathcal{C}_{k-1}$ are not prerequisites of $\mathcal{C}_k$. For example, FI (file) is to the left of IN (instruction) in the list in (4.1), but it is not a prerequisite of IN. When such cases are taken into consideration, the number of valid expressions would be bigger, but is still much smaller than $2^N$.

In implementing the ITS, each state expression is coded as one or more unsigned long integer of 64 bits, depending on the number of concepts. Each concept condition is coded as a bit, with value 1 for "understand", and value 0 for "not understand". In this way, we can use an unsigned long integer to code an expression of up to 64 concepts. If the number of concepts is larger than 64, we can use two or more words for a state expression.

Our state definition is different from those in the existing work, in which states are commonly defined by a number of "features", such as correctness, certainty, concept repetition, frustration, percent correct, etc [27]. Our definition depicts the student's study states. It has the advantages: In tutoring a student, information about what the student understands, i.e. the study states, is essential in deciding a tutoring strategy. Since the information is available in each state, the system may choose the most suitable answer to the student's question based on the locally available information. Also, a state defined in this way is Markovian. The Markov property makes the state definition suitable for RL.

## 4.3 Actions

In a tutoring session, asking and answering questions are the primary actions of the student and system. Other actions are those for confirmation, etc. Figure 4.2 shows some system and user actions in a tutoring dialogue session. Note that the knowledge of software in our examples is simplified so that we can better focus on the techniques for system development.

We classify actions in the ITS into *student actions* and *system actions*. Student

```
m1:  What would you like to learn?
h1:  What is a database?
m2:  A database is a collection of interrelated computer files and
     a set of application programs written in a query language.
h2:  But what is a query language?
m3:  A query language is a high-level language mainly for requesting
     data.
h3:  I see.
m4:  Now I return to database.
     A database is a collection of interrelated computer files and
     a set of application programs written in a query language.
     Make sense?
h4:  Yes.
m5:  Other questions?
h5:  What is a human-computer interface?
...
```

Figure 4.2: System and student actions in a tutoring session. m - machine (system), h - human (student).

actions mainly includes the actions of asking questions about concepts. Asking "what is a database?" is such an action. Student actions can also be "Tell me about machine language", "I need to know about computer file", etc. Each student action involves only one concept. In the following discussion, we denote a student action of asking about concept $C$ by $[C]$. For example, when $C$ represents "database", $[C]$ is an action of asking about "database". We use [ ] to denote a student action that is not a question about a concept, like "please continue" and "I am done". We code a student action using the same number of unsigned long integers as a state expression. The coding is similar to that for state expressions, in which each concept is represented by a bit and the concepts are topologically sorted. In the coding of a student action, one bit is assigned "1" representing the concept that is asked.

The system actions mainly include the actions of answering questions about concepts, like "A database is a collection of interrelated computer files and a set of application programs written in a query language". We use $\{C\}$ to denote a system

action of explaining $C$. We use { } to denote a system action that does not explain a concept, like "how can I help you?"

Quite often, the system can answer $[C]$ directly by taking the action of $\{C\}$. However, sometimes to answer a question, the system has to "make up" some prerequisite knowledge. For example, before answering a question about "database", the system has to explain "query language" and "computer file" if the student does not understand them. Let's use $C_l$ represent "database", $C_j$ represent "query language", and $C_k$ represent "computer file" $(1 \leq j < k < l)$, and assume $j < k$. Subscripts $j$, $k$, $l$ are indexes of the concepts in the state expressions, which are topologically sorted. We express such actions as $\{C_jC_kC_l\}$, which specifies that the system explains $C_j$, then $C_k$, and then $C_l$. It expains $C_j$ and $C_k$ in order to eventually explain $C_l$. Similarly, if the system first explains "high-level language" in order to explain "query language", the actions are expressed as $\{C_iC_jC_kC_l\}$, where $C_i$ represents "high-level language" $(1 \leq i < j < k < l)$. In the following discussion, we refer to such a sequence of actions for answering a question as a *answer path*.

For an answer path like $\{C_iC_jC_kC_l\}$, the system explains the concepts from left to right. Since the concepts are topologically sorted, a concept is explained after all the concepts on its left hand side (i.e. the prerequisites) are explained.

## 4.4 State Transitions by Actions

In the ITS architecture, system actions causes state transitions, while student action are used as observations. We will denote a system action by $a$, and denote a student action by $o$.

In a state $s \in S$, any system action can be taken. A system action can transit the state into different states. In the following, we will take the state of $(\neg C_1, \neg C_2, ..., \neg C_N)$ as an example, which indicates that the student understands nothing about the subject. Assume the system action in this state is a greeting like "Hi. What would you

like to learn?" denoted by { }. This action causes a state transition. Since states are not observable, we can only estimate the new state. The new state is estimated by observing the student action. Assume the student asks question $[C_l]$. The most probable new state may be one of

- $(\neg C_1 \neg C_2 ... \neg C_{l-1} \neg C_l \neg C_{l+1} ... \neg C_N)$: The student does not understand $C_l$ and all of its prerequisites;
- $(\surd C_1 \neg C_2 ... \neg C_{l-1} \neg C_l \neg C_{l+1} ... \neg C_N)$: The student does not understand $C_l$ and all of its prerequisites except for $C_1$;
- ...;
- $(\surd C_1 \surd C_2 ... \surd C_{l-1} \neg C_l \neg C_{l+1} ... \neg C_N)$: The student does not understand $C_l$ but understands all of its prerequisites.

(Keep in mind that the concepts are topologically sorted.) When $a$ is taken in $s$, there is a probability for transition into new state $s'$, denoted by $P(s'|s,a)$. For each possible new state $s'$ caused by $a$, there is a probability to observe student action $o$, denoted by $P(o|a,s')$.

In the new state, similarly, any system action can be taken, and the action taken may transit the state into different states.

In a state, for a given student question, there is a set of "most suitable" system actions. For example, if the user question is $[C_l]$, and the new state is $(\surd C_1 \neg C_2 ... \neg C_{l-1} \neg C_l \neg C_{l+1} ... \neg C_N)$, $\{C_2\}$ is one of the most suitable system actions when $C_2$ is a prerequisite of $C_l$. We do not need to teach $C_1$ because the student already understands it. We should not omit $C_2$ because the student does not understand $C_2$, which is eventually needed to understand $C_l$.

In an application, if states are completely observable, it is relatively easy for the system to choose the most suitable actions to take. However, if states are not observable, or not completely observable, choosing the most suitable action in a state is a challenging task. We use the technique of POMDP to address this challenge.

## 4.5 Teaching strategy as policy trees

As mentioned before, a concept may have prerequisites. When answering a question about a concept, an ITS may directly explain the concept, or it may start with one of the prerequisites to make up the knowledge that is needed for the student to understand the concept in question. In this thesis, *teaching strategy* is used to select of the starting concept in answering a question. The teaching strategy largely determines student satisfaction and teaching efficiency. In the following, we discuss more about teaching strategies, and the problems that a poor strategy may cause.

The same student question can be answered in different ways. Assume the student question is $[C_k]$ and $C_k$ has prerequisites $C_1$, $C_2$, ..., $C_{k-1}$. A possible system action is to teach $C_k$ directly, without making up any prerequisites. The second possible action is to start with $C_1$, then teach $C_2$, until $C_k$. The third actio is to start with $C_2$, then teach $C_3$, until $C_k$, and so on. For example, when asked about "database", the agent may explain what a database is, without making up any prerequisite (like `m2` in Figure 4.2). A disadvantage is that the student may become frustrated and has to ask about many prerequisites. Another answer is to start with a very basic prerequisite. For example, the agent starts with "binary digit" when answering about "database". A disadvantage of this answer is low efficiency and the student may become impatient. When answering a student question, the system should be able to choose a good answer which starts with "right" prerequisite if the concept in question has prerequisites. In the example in Figure 4.2, a good answer is to start with "query language".

In our technique, we model the teaching strategy as the policy of the learning agent. In POMDP, policy trees can be used to simplify the process of POMDP solving, that is, the process to find the optimal policy. In a policy tree, a node represents an action and an edge represents an observation. When executing a policy tree, the system first takes the action at the root node, then depending on the observation, takes the action at the node that is connected by the edge representing the observa-

tion, and so on.

A policy is comprised of a set of policy trees. When we use POMDP to solve a problem, we select the policy tree that maximizes the value function. For different belief state $b$, we choose different policy trees to solve the problem. The calculation for policy selection is given in Eqn (3.22). The equation is repeated here for the convenience of discussion:

$$\pi(b) = \hat{p} = \arg\max_{p \in \mathcal{P}} V_p(b).$$

In our technique, we use the method of policy trees. In a policy tree, a node is associated with a system action (explaining a concept) and an edge is associated with a student action (treated as an observation). A student action may be asking a question, accepting or rejecting a system action.

For each concept, we create a set of policy trees to answer student questions about the concept. Let the concept in question be $C_l$. For each *direct* prerequisite $C_k$, we create a policy tree $p$ with $C_k$ being the root. In the policy tree, there is one or more paths $C_k, ..., C_l$, which are *answer paths* for student question $[C_l]$. When the student asks a question about $C_l$, we select the optimal policy tree $p$ that contain the most suitable answer path, based on the student's current study state.

## 4.6 Value function and reward function

To apply Eqn (3.22) to select the optimal policy tree for student question $[C_l]$, we use value function $V_p(b)$. The optimal policy tree for answering $[C_l]$ is the $\hat{p}$ that maximizes $V_p$ for given $b$ that represents the student's current study state. (In the following, for the convenience of discussion, we repeated some important equations.) $V_p(b)$ is defined in (3.20):

$$V_p(b) = b \cdot V_p.$$

where $V_p$ is a vector defined in (3.19):

$$V_p = [V_p(s^1), V_p(s^2), ..., V_p(s^N)].$$

where $N$ is the number of physical states. For a given physical state $s$, $V_p(s)$ is defined in (3.15):

$$V_p(s) = \mathcal{R}(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \sum_{o \in O} P(o|a,s') V_{p(o)}(s')$$

where $a$ is the root action of policy tree $p$, $\mathcal{R}(s,a)$ is the expected immediate reward after the agent takes $a$ in $s$, and $p(o)$ is the subtree of $p$ that is connected with the root by the edge labeled $o$.

In our technique, we define reward function as $\rho : S \times A \rightarrow \Re$. That is, the reward function is of the form $\rho(s,a)$. Let $\{C_l\}$ be system action $a$, $\wp_l$ be the set of all the direct and indirect prerequisite concepts of $C_l$, and $s$ be represented by state expression $(\mathcal{C}_1\mathcal{C}_2...\mathcal{C}_N)$. We define the reward function as

$$\rho(s,a) = \begin{cases} r' & \text{if } (a = \{C_l\}) \wedge \forall \mathcal{C} \in \wp_l (\mathcal{C} = \surd C) \wedge (\mathcal{C}_l = \neg C_l) \\ r'' & \text{otherwise} \\ 0 & \text{if } a = \{\ \}, \end{cases} \tag{4.2}$$

where $r'$ and $r''$ are two scalar values and $r' > r''$. When the agent takes system action $\{C_l\}$ in state $s$, if in the state expression of $s$ all the direct and indirect prerequisites of $C_l$ are in "understand" condition and $C_l$ is "not understand", the agent receives $r'$, otherwise receives $r''$.

## 4.7 An example

In the following, we use a subset of the concepts in software basic to create a miniature ITS to show some major components in the ITS based on POMDP. The concepts are "binary digit" (BD), "bit" (BI), "instruction" (IN), "programming language" (PL), and "machine language" (ML). PL and BD are prerequisites of ML. IN is a prerequisite of PL. Assume we develop an ITS for tutoring the four concepts. Same as

before, when numbering the states, actions, and observations, we place the numbers as superscripts, to avoid confusion with time steps. (Note we still use $s'$ to denote the state in the next time step.)

The valid physical states are:

$s^1 : (\neg BD \neg BI \neg IN \neg PL \neg ML)$,
$s^2 : (\surd BD \neg BI \neg IN \neg PL \neg ML)$,
$s^3 : (\surd BD \surd BI \neg IN \neg PL \neg ML)$,
$s^4 : (\surd BD \surd BI \surd IN \neg PL \neg ML)$,
$s^5 : (\surd BD \surd BI \surd IN \surd PL \neg ML)$,
$s^6 : (\neg BD \neg BI \surd IN \neg PL \neg ML)$,
$s^7 : (\neg BD \neg BI \surd IN \surd PL \neg ML)$,
$s^8 : (\surd BD \neg BI \surd IN \surd PL \neg ML)$,
$s^9 : (\surd BD \surd BI \surd IN \surd PL \surd ML)$.

The number of valid states is 9, smaller than $2^N$, where $N = 5$. The students actions are

$o^1 : [BD]$, $o^2 : [BI]$, $o^3 : [IN]$, $o^4 : [PL]$, $o^5 : [ML]$, $o^6 : [\,]$, $o^7 :$ [rejection],

where [ ] denotes an acceptance action, and [rejection] denotes a rejection action.

The system actions are

$a^1 : \{BD\}$, $a^2 : \{BI\}$, $a^3 : \{IN\}$, $a^4 : \{PL\}$, $a^5 : \{ML\}$, $a^6 : \{\,\}$,

where $\{C\}$ denotes a description/explanation about concept $C$, and empty { } denotes a greeting or "how can I help you?" The reward function is

$\rho(s^1, a^1) = r'$, $\rho(s^1, a^2) = r''$, $\rho(s^1, a^3) = r''$, $\rho(s^1, a^4) = r''$, $\rho(s^1, a^5) = r''$, $\rho(s^1, a^6) = 0$,
$\rho(s^2, a^1) = r''$, $\rho(s^2, a^2) = r'$, $\rho(s^2, a^3) = r'$, $\rho(s^2, a^4) = r''$, $\rho(s^2, a^5) = r''$, $\rho(s^2, a^6) = 0$,
$\rho(s^3, a^1) = r''$, $\rho(s^3, a^2) = r''$, $\rho(s^3, a^3) = r'$, $\rho(s^3, a^4) = r''$, $\rho(s^3, a^5) = r''$, $\rho(s^3, a^6) = 0$,
$\rho(s^4, a^1) = r''$, $\rho(s^4, a^2) = r''$, $\rho(s^4, a^3) = r''$, $\rho(s^4, a^4) = r'$, $\rho(s^4, a^5) = r''$, $\rho(s^4, a^6) = 0$,

$\rho(s^5, a^1) = r''$, $\rho(s^5, a^2) = r''$, $\rho(s^5, a^3) = r''$, $\rho(s^5, a^4) = r''$, $\rho(s^5, a^5) = r'$, $\rho(s^5, a^6) = 0$,
$\rho(s^6, a^1) = r'$, $\rho(s^6, a^2) = r''$, $\rho(s^6, a^3) = r''$, $\rho(s^6, a^4) = r'$, $\rho(s^6, a^5) = r''$, $\rho(s^6, a^6) = 0$,
$\rho(s^7, a^1) = r'$, $\rho(s^7, a^2) = r''$, $\rho(s^7, a^3) = r''$, $\rho(s^7, a^4) = r''$, $\rho(s^7, a^5) = r''$, $\rho(s^7, a^6) = 0$,
$\rho(s^8, a^1) = r''$, $\rho(s^8, a^2) = r'$, $\rho(s^8, a^3) = r''$, $\rho(s^8, a^4) = r''$, $\rho(s^8, a^5) = r'$, $\rho(s^8, a^6) = 0$,
$\rho(s^9, a^1) = r''$, $\rho(s^9, a^2) = r''$, $\rho(s^9, a^3) = r''$, $\rho(s^9, a^4) = r''$, $\rho(s^9, a^5) = r''$, $\rho(s^9, a^6) = 0$.

For the values assigned to the state-action pairs, please see Eqn (4.2).

In a state, the agent takes a (system) action. The action transits the current state into a new state. After the transition, the agent receives a reward. Then the student takes a (student) action. The student action is observed by the agent. The agent uses the observed student action to estimate the new state.

Th transition probabilities are

$P(s^1|s^1, a^1)$, $P(s^2|s^1, a^1)$, $P(s^3|s^1, a^1)$, $P(s^4|s^1, a^1)$, $P(s^5|s^1, a^1)$, $P(s^6|s^1, a^1)$, $P(s^7|s^1, a^1)$, $P(s^8|s^1, a^1)$, $P(s^9|s^1, a^1)$,
$P(s^1|s^2, a^1)$, $P(s^2|s^2, a^1)$, $P(s^3|s^2, a^1)$, $P(s^4|s^2, a^1)$, $P(s^5|s^2, a^1)$, $P(s^6|s^2, a^1)$, $P(s^7|s^2, a^1)$, $P(s^8|s^2, a^1)$, $P(s^9|s^2, a^1)$,
...
$P(s^1|s^9, a^1)$, $P(s^2|s^9, a^1)$, $P(s^3|s^9, a^1)$, $P(s^4|s^9, a^1)$, $P(s^5|s^9, a^1)$, $P(s^6|s^9, a^1)$, $P(s^7|s^9, a^1)$, $P(s^8|s^9, a^1)$, $P(s^9|s^9, a^1)$,
...
$P(s^1|s^1, a^6)$, $P(s^2|s^1, a^6)$, $P(s^3|s^1, a^6)$, $P(s^4|s^1, a^6)$, $P(s^5|s^1, a^6)$, $P(s^6|s^1, a^6)$, $P(s^7|s^1, a^6)$, $P(s^8|s^1, a^6)$, $P(s^9|s^1, a^6)$,
$P(s^1|s^2, a^6)$, $P(s^2|s^2, a^6)$, $P(s^3|s^2, a^6)$, $P(s^4|s^2, a^6)$, $P(s^5|s^2, a^6)$, $P(s^6|s^2, a^6)$, $P(s^7|s^2, a^6)$, $P(s^8|s^2, a^6)$, $P(s^9|s^2, a^6)$,
...
$P(s^1|s^9, a^6)$, $P(s^2|s^9, a^6)$, $P(s^3|s^9, a^6)$, $P(s^4|s^9, a^6)$, $P(s^5|s^9, a^6)$, $P(s^6|s^9, a^6)$, $P(s^7|s^9, a^6)$, $P(s^8|s^9, a^6)$, $P(s^9|s^9, a^6)$,

Many state transition probabilities are zeros, which indicate impossible transitions. For $s^i$ and $s^j$, if $\mathcal{C}_l = \surd C_l$ in $s^j$ and $\mathcal{C}_l = \neg C_l$ in $s^i$ $(1 \leq l \leq 5)$, then $\forall k(P(s^i|s^j, a^k) = 0)$. The reason is that no action can be taken to change a student from a state of understanding a concept into a state of not understanding the same concept. The non-zero transition probabilities are first initialized by using training data, and then improved during system-user interactions.

The observation probabilities are

$P(o^1|a^1, s^1)$, $P(o^1|a^1, s^2)$, $P(o^1|a^1, s^3)$, $P(o^1|a^1, s^4)$, $P(o^1|a^1, s^5)$, $P(o^1|a^1, s^6)$, $P(o^1|a^1, s^7)$, $P(o^1|a^1, s^8)$, $P(o^1|a^1, s^9)$,
$P(o^1|a^2, s^1)$, $P(o^1|a^2, s^2)$, $P(o^1|a^2, s^3)$, $P(o^1|a^2, s^4)$, $P(o^1|a^2, s^5)$, $P(o^1|a^2, s^6)$, $P(o^1|a^2, s^7)$, $P(o^1|a^2, s^8)$, $P(o^1|a^2, s^9)$,
...
$P(o^1|a^5, s^1)$, $P(o^1|a^5, s^2)$, $P(o^1|a^5, s^3)$, $P(o^1|a^5, s^4)$, $P(o^1|a^5, s^5)$, $P(o^1|a^5, s^6)$, $P(o^1|a^5, s^7)$, $P(o^1|a^5, s^8)$, $P(o^1|a^5, s^9)$,
...
$P(o^7|a^1, s^1)$, $P(o^7|a^1, s^2)$, $P(o^7|a^1, s^3)$, $P(o^7|a^1, s^4)$, $P(o^7|a^1, s^5)$, $P(o^7|a^1, s^6)$, $P(o^7|a^1, s^7)$, $P(o^7|a^1, s^8)$, $P(o^7|a^1, s^9)$,
$P(o^7|a^2, s^1)$, $P(o^7|a^2, s^2)$, $P(o^7|a^2, s^3)$, $P(o^7|a^2, s^4)$, $P(o^7|a^2, s^5)$, $P(o^7|a^2, s^6)$, $P(o^7|a^2, s^7)$, $P(o^7|a^2, s^8)$, $P(o^7|a^2, s^9)$,
...
$P(o^7|a^5, s^1)$, $P(o^7|a^5, s^2)$, $P(o^7|a^5, s^3)$, $P(o^7|a^5, s^4)$, $P(o^7|a^5, s^5)$, $P(o^7|a^5, s^6)$, $P(o^7|a^5, s^7)$, $P(o^7|a^5, s^8)$, $P(o^7|a^5, s^9)$.

Many observation probabilities are zeros, which indicate impossible observations. Let $o^l = [C_l]$ $(1 \leq l \leq 5)$. If $\mathcal{C}_l = \surd C_l$ in $s^i$, then $\forall k(P(o^l|a^k, s^i) = 0)$. The reason is that no action can be taken to make a student to ask a question about a concept that he/she already understands. The non-zero observation probabilities are first initialized by using training data, and then improved during system-user interactions.

## 4.8 Policy initialization

From (3.22), (3.20), (3.19), and (3.15), we can see that a policy is defined by the $V$ function, and the $V$ function is defined by $\mathcal{R}(s,a)$, $P(s'|s,a)$, and $P(o|a,s')$. The creation of $\mathcal{R}(s,a)$, $P(s'|s,a)$, and $P(o|a,s')$ is the primary task in policy initialization. $\mathcal{R}(s,a)$ is the expected reward after the agent takes $a$ in $s$. Since we define the reward function as $S \times A \rightarrow \Re$ (see Eqn (4.2)), we have $\mathcal{R}(s,a) = \rho(s,a)$, policy initialization mainly involves creating $P(s'|s,a)$ and $P(o|a,s')$. We will see later that updating them is also the primary task in policy improvement.

$P(s'|s,a)$ is defined as

$$P(s'|s,a) = P(s_{t+1} = s'|s_t = s, a_t = a) \tag{4.3}$$

where $t$ denotes a time step, and $s'$ is the new state at time step $t+1$. $P(o|a,s')$ is defined as

$$P(o|a,s') = P(o_{t+1} = o|a_t = a, s_{t+1} = s') \tag{4.4}$$

where $t$ and $t+1$ are the same as (4.3).

To initialize $P(s'|s,a)$ and $P(o|a,s')$ for our algorithm, we create action sequences as training data. A sequence consists of system actions along with the states where the actions are taken, the student actions, and the new states. The data are represented as tuples:

$$...(\breve{s}_t, a_t, o_t, \breve{s}_{t+1})(\breve{s}_{t+1}, a_{t+1}, o_{t+1}, \breve{s}_{t+2})... \tag{4.5}$$

where the $a$ denoted a system action, and $o$ denotes a student action since the agent treats a student action as an observation.

Let $s$ be $\breve{s}_t$, and $s'$ be $\breve{s}_{t+1}$. $P(s'|s,a)$ can be initialized as

$$P(s'|s,a) = \frac{\text{Count(transition from } s \text{ to } s' \text{ when } a \text{ is taken in } s)}{\text{Count}(a \text{ is taken in } s)}. \tag{4.6}$$

$P(o|a,s')$ can be initialized as

$$P(o|a,s') = \frac{\text{Count}(a \text{ is taken, } s' \text{ is perceived, and } o \text{ is observed})}{\text{Count}(a \text{ is taken and } s' \text{ is perceived})}. \tag{4.7}$$

In (4.6) and (4.7) the counts and sum are from the training data in the form of (4.5). We use the Lidstone estimate [20] to deal with the data sparsity problem.

## 4.9 Policy execution

As we have seen that in the policy initialization, probabilities $P(s'|s,a)$ and $P(o|a,s')$ are assigned their initial values. After the initialization is completed, the agent can calculate $V$ for policy selection and evaluate the new belief state $b'$.

When the ITS starts to teach a new student, the belief state $b$ is initialized as $b_0 = (1/M, 1/M, ..., 1/M)$, where $M$ is the number of valid states. At the beginning, the agent has no information about the student's study state. We thus assign equal probabilities to all the physical states. That is, $b(s^i) = 1/M$ for all the $i$.

The agent first greets the student. For example, in the sample system showed above, the agent takes $a^6$. After the greeting, the student asks a question. The agent treats the student action (question) as an observation $o$, and calculates the new belief state, using Eqns (3.11) and (3.13), which are repeated here for convenience:

$$P(o|a) = \sum_{s\in S} b(s) \sum_{s'\in S} P(s'|s,a)P(o|a,s'),$$

and

$$b'(s') = \sum_{s\in S} b(s)P(s'|s,a)P(z|a,s')/P(o|a).$$

The agent then calculates $V_p(s)$ for policy trees and physical states using Eqn (3.15):

$$V_p(s) = \mathcal{R}(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \sum_{o\in O} P(o|a,s')V_{p(o)}(s').$$

And then the agent calculates $V_p(b)$ for the current belief state $b$ and chooses the optimal $\hat{p}$ to execute, using Eqns (3.18) and (3.22):

$$V_p(b) = \sum_{s\in S} b(s)V_p(s),$$

and

$$\pi(b) = \hat{p} = \arg\max_{p \in \mathcal{P}} V_p(b).$$

After the system action, the student takes another action, which is treated as an observation. The agent uses this observation, along with the probabilities of $P(s'|s,a)$ and $P(o|a,s')$, to evaluate the next belief state. Based on the belief state, it chooses the next optimal $p$ to execute, and so on.

## 4.10 Policy improvement

Policy improvement updates $P(s'|s,a)$ and $P(o|a,s')$, so that belief states can model physical states better. The objective of policy improvement is to enable the agent to choose more understandable and more efficient answers to student questions.

In reinforcement learning, the learning agent improve its policy through the interaction with the environment, and the policy improvement is conducted when the agent applies the policy to solve problems.

In our technique, we use a delayed updating method for policy improvement. In this method, the current policy is fixed for a certain number of tutoring sessions. In the tutoring sessions, system and student actions are recorded. A tutoring session starts when the system greets the student and the student asks a question about a concept, and ends when the system answers a related question and the student indicates his/her understanding of the concept asked. After the tutoring sessions, while the policy continues to work, information about the recorded actions is processed, and the transition probabilities and observation probabilities are updated. When the improvement is completed, the updated probabilities replace the current ones and are used for choosing system actions, then they are updated again after a certain number of tutoring sessions, and so on.

The recorded data for policy improvement are sequences of tuples. In the tuples, we use $a$ to denote a system action and $o$ to denote a student action. The recorded

data are

$$...(\hat{s}_t, a_t, o_t, \hat{s}_{t+1})(\hat{s}_{t+1}, a_{t+1}, o_{t+1}, \hat{s}_{t+2})... \tag{4.8}$$

where $\hat{s}_i$ is the most probable physical state in $b_i$ $(i = 1, ..., t, t+1, ...)$ At time step $i$, the agent believes that it is most likely in $\hat{s}_i$. In the following, we call $\hat{s}$ the *believed physical state*.

The recorded tuple sequences are modified for updating the probabilities. We modify believed physical state $\hat{s}_{t+1}$ by using student action $o_t$. Here we use tuple $(\hat{s}_t, a_t, o_t, \hat{s}_{t+1})$ to explain the modification. Assume $o_t = [C_l]$, and the expression of $\hat{s}_{t+1}$ is $(...\surd C_j \surd C_k \neg C_l...)$ where $C_j$ and $C_k$ are prerequisites of $C_l$. That is, the student asks a question about $C_l$, and to the agent, the student is in a study state of not understanding $C_l$ but understanding $C_j$ and $C_k$. If in the subsequent tuples in the same recorded tutoring session there are student actions of $o_{t+1} = [C_j]$ and $o_{t+2} = [C_k]$, we modify the expression of $\hat{s}_{t+1}$ into $(...\neg C_j \neg C_k \neg C_l...)$, which is for a state in which the student does not understand the three concepts. This is a different state. We thus modify the tuple into $(\hat{s}_t, a_t, o_t, \breve{s}_{t+1})$, where $\breve{s}_{t+1}$ is the state represented by $(...\neg C_j \neg C_k \neg C_l...)$. In the following, we use $\breve{s}$ for the states in the modified tuples.

After the modification, the tuple sequences for updating the probabilities become

$$...(\breve{s}_t, a_t, o_t, \breve{s}_{t+1})(\breve{s}_{t+1}, a_{t+1}, o_{t+1}, \breve{s}_{t+2})... \tag{4.9}$$

From (4.9), we derive sequence

$$...(\breve{s}_t, a_t, \breve{s}_{t+1})(\breve{s}_{t+1}, a_{t+1}, \breve{s}_{t+2})... \tag{4.10}$$

for updating $P(s'|s, a)$, and derive sequence

$$...(a_t, o_t, \breve{s}_{t+1})(a_{t+1}, o_{t+1}, \breve{s}_{t+2})... \tag{4.11}$$

for updating $P(o|a, s')$.

$P(s'|s, a)$ is updated as

$$P(s'|s, a) = C_1/(C_2 + C_4) + C_3/(C_2 + C_4) \tag{4.12}$$

where

- $C_1$ is the accumulated count of tuples $(s, a, s')$ in which $s = \breve{s}_t$, $a = a_t$ and $s' = \breve{s}_{t+1}$ in initialization and all the previous updates.
- $C_2$ is the accumulated count of tuples $(s, a, *)$ in which $s = \breve{s}_t$, $a = a_t$, and $*$ is any state in initialization and all the previous updates.
- $C_3$ is the count of tuples $(s, a, s')$ in which $s = \breve{s}_t$, $a = a_t$ and $s' = \breve{s}_{t+1}$ $a_t = a$ and $\breve{s}_{t+1} = s'$ in the current update.
- $C_4$ is the count of tuples $(s, a, *)$ in which $s = \breve{s}_t$, $a = a_t$, and $*$ is any state and $a_t = a$ in the current update.

$P(o|a, s')$ is updated in the same way.

# Chapter 5

# Experiments and Analysis of Results

## 5.1 Implementation

### 5.1.1 An overview

An experimental system is developed in C to implemented the technique. It is an ITS tutoring the basic level knowledge of software. Example concepts taught by the system are listed in Appendice A and B.

The ITS is designed as an interactive system: a student communicates with the ITS one at a time to ask questions and get answers. Every time, the student asks a question, the ITS answers the question. Based on the previous question/answer, the student may ask another question which could be a new one or to know a prerequisite for understanding a previous question. The ITS answers the new question, ... and so on, until the student finishes his questions.

In the ITS, the device used for student input is the keyboard, and the device for system output is the screen.

### 5.1.2 The system components

The main system components include an interpreter, an agent module, and a set of databases.

The interpreter parses and analyzes a student input, and identifies the concept asked if the input is a question.

The main databases are:

- The database of system actions: Most of the system actions are answers to student questions asking concepts. Human readable forms of the system actions can be found in Appendix A.
- The database of policy trees: This database contains the policy trees, grouped by the concepts they are used to answer/describe.
- The database of transition probabilities: This database contains state transition probabilities of $P(s'|s, a)$. The probabilities are updated in each delayed update.
- The database of observation probabilities: This database contains observation probabilities of $P(o|a, s')$. The probabilities are updated in each delayed update.
- The database of rewards: This database contains rewards of $\mathcal{R}(s, a)$.

The agent module performs the functionality of the learning agent. It accesses the databases to make decisions, and responds to the students. It also updates the databases to improve the teaching strategy.

### 5.1.3 Policy creation, execution and update

In our tutoring system, we use policy trees to simplify the POMDP decision process in order to find the optimal policy. With a policy tree, the system takes actions from top (root node) to bottom depending on the observations (student actions) at each node. For each concept, we define a set of policy trees in order to answer the student

questions about the concept. For each question $C_i$, its direct prerequisite is $C_{i-1}$, a policy tree $p$ is defined with $C_{i-1}$ as root at top.

A policy consists of the policy trees, transition probabilities $P(s'|s,a)$, observation probabilities $P(o|a,s')$, and rewards $\mathcal{R}(s,a)$. In order to solve a problem, for different belief state $b$, we calcuate the $V$ function of $P(s'|s,a)$, $P(o|a,s')$ and $\mathcal{R}(s,a)$ to choose different policy trees.

The initial policy is created by using on (4.5), (4.6) and (4.7). The database values are initialized.

The belief state $b$ is initialized with equal probabilities to all the physical states. A global variable is used to store the current belief state.

Before the agent takes an action, it accesses the value of the $b$ variable, accesses the databases of $P(s'|s,a)$, $P(o|a,s')$ and $\mathcal{R}(s,a)$, and calculates (3.17), (3.18) and (3.22) to get the optimal $\hat{p}$ for the system to take action. When the student sees the system action, the student takes another action $o'$. With the observation $o'$, the agent accesses the databases of $P(s'|s,a)$, $P(o|a,s')$, and $\mathcal{R}(s,a)$ to calculate (3.11) and (3.13) to get next belief state $b'$, and then choose the next $\hat{p}$, and so on.

The agent improves the policy interactively with delayed updating method. For our ITS system, to update the policy is to update both transition probabilities $P(s'|s,a)$ and observation probabilities $P(o|a,s')$. The agent calculates (4.12) to get new values of both $P(s'|s,a)$ and $P(o|a,s')$. Then the two databases of the probabilities are updated.

## 5.2 Experiment Design

We use a two-sample *t*-test method to evaluate how well the optimized teaching strategy can improve the teaching performance of an ITS. The test method is the *independent-samples t-test*. In the following, we first briefly describe the experimental design.

We partition the experiment participants into two groups: Group 1 and Group 2. The participants in Group 1 study with the ITS when the optimized teaching strategy is not used, while the participants in Group 2 study with the ITS when the optimized teaching strategy is used for choosing answers to questions.

In the experiment, the performance parameter is the *rejection rate* of system actions. In a tutoring session, a student asks a question about a concept, the system takes actions to answer the question directly or explain some prerequisites of the concept, the student may ask about a prerequisite concept for understanding the concept, the system may answer the second question and so on, until the student understands the concept in question. In the example in Figure 4.2, actions from `m1` through `h4` form a tutoring session.

A tutoring session is a sequence of $(a, o)$ pairs where $a$ is a system action and $o$ is a student action following $a$. Assume $a = \{C_l\}$, and $o = [C_k]$. If $C_k$ is a prerequisite of $C_l$, we say that the system action $a$ is rejected by the student. In the example in Figure 4.2, system action `m2` is rejected by `h2`. A system action may also be rejected explicitly when the student says "I already know this".

*Rejection rate* is defined as

$$\text{rejection rate} = \frac{\text{number of system actions rejected in the tutoring session}}{\text{total number of system actions in the tutoring session}}. \quad (5.1)$$

For each participant, we calculate the mean rejection rate. For the two groups, we calculate means $\bar{X}_1$ and $\bar{X}_2$. Sample mean $\bar{X}_1$ is used to represent population mean $\mu_1$. and $\bar{X}_2$ represents $\mu_2$. We must determinate if the difference between the sample means reflects sampling error before we make the decision.

The alternative hypothesis is:

$$H_a : \mu_1 - \mu_2 \neq 0$$

$H_a$ implies the means from the two groups represent different populations. That is, the optimized teaching strategy changes the teaching performance of the ITS by reducing the rejection rate.

The null hypothesis is

$$H_0 : \mu_1 - \mu_2 = 0$$

$H_0$ implies both samples represent the same population. That is, the optimized teaching strategy does not change the teaching performance.

In order to test $H_0$, we use $t_{obt}$ to determinate where the difference between means lies on this sampling distribution. First, we estimate population variance $s_i^2$ for each group

$$s_i^2 = \frac{\sum X^2 - \frac{(\sum X)^2}{n}}{n-1}, \tag{5.2}$$

where $i$ is 1 or 2, $n$ is the number of rates in a group, and $X$ is a rejection rate in that group. (One rate is for a participant.) In this way, we can get $s_1^2$ and $s_2^2$, each is an estimate of the population variance.

Based on $s_1^2$ and $s_2^2$, we calculate the pooled variance:

$$s_{pool}^2 = \frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{(n_1-1)+(n_2-1)}, \tag{5.3}$$

where $n_1$ is the number of rates in Group 1 and $n_2$ is the number of rates in Group 2.

Using the pooled variance, we calculate the standard error of the difference:

$$s_{\bar{X}_1-\bar{X}_2} = \sqrt{(s_{pool}^2)(\frac{1}{n_1}+\frac{1}{n_2})} \tag{5.4}$$

Next, we calculate the independent-samples $t_{obt}$

$$t_{obt} = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{s_{\bar{X}_1-\bar{X}_2}}, \tag{5.5}$$

where $\bar{X}_1$ and $\bar{X}_2$ are our sample means, $s_{\bar{X}_1-\bar{X}_2}$ is computed as (5.4), and the value of $\mu_1 - \mu_2$ is the difference specified by the null hypothesis. This value is always 0.

In order to determine if $t_{obt}$ is significant, we can compare it with $t_{crit}$. $t_{crit}$ can be looked up from a $t$-table by using the degrees of freedom:

$$bf = (n_1-1)+(n_2-1), \tag{5.6}$$

where $n_1$ and $n_2$ are the numbers of participants (i.e. numbers of rejection rates) in the two groups.

Finally, we compare $t_{obt}$ and $t_{crit}$. If $t_{obt}$ is within the range defined by $t_{crit}$, we accept $H_0$ and reject $H_a$. If $t_{obt}$ is outside the range defined by $t_{crit}$, we accept $H_a$ and reject $H_0$.

## 5.3 Experiments

In our experiment, there were totally 30 participants (called students in the following), divided randomly into two groups of equal sizes. The participants are ordinary people who know how to use a computer and some application programs like Web browser, MS office, etc. but do not have formal training in software development and did not take a course on software.

Each participant studies with the ITS for approximately 45 minutes, asking questions by typing and seeing the answers on the screen.

Group 1 was tested without the improved teaching strategy. When the ITS taught a student, POMDP was running and calculated the new belief state after each system action. The belief states are used to get $\hat{s}$ in the recorded data for updating the probabilities. The choice of starting point to answer a question is random. That is, to answer a question about a concept, the system randomly choose a prerequisites of the concept in question, or directly explains the concept. The initialized probabilities are fixed, not updated. The student and system actions are recorded. Later they are read to improve the teaching strategy.

Group 2 was tested with the improved teaching strategy. To answer a question, the system uses the improved strategy. The system continues to improve the strategy using the delayed update method. At the beginning of teaching a student, the strategy is initialized as the one improved by using the Group 1 data.

The system performance is measured in terms of the rate of system actions rejected by the students. The raw experimental data (recorded rejection rates) of the participants in the two groups are listed in Table 5.1.

Table 5.1: Rejection rates of Group 1 and Group 2 participants.

| Group 1 | | Group 2 | |
|---|---|---|---|
| Student | Rejection Rate | Student | Rejection Rate |
| 1 | 0.4286 | 1 | 0.2143 |
| 2 | 0.6429 | 2 | 0.0714 |
| 3 | 0.5172 | 3 | 0.3571 |
| 4 | 0.5714 | 4 | 0.2143 |
| 5 | 0.4286 | 5 | 0.3571 |
| 6 | 0.5812 | 6 | 0.4286 |
| 7 | 0.5201 | 7 | 0.2123 |
| 8 | 0.6429 | 8 | 0.2453 |
| 9 | 0.6206 | 9 | 0.0714 |
| 10 | 0.8571 | 10 | 0.2571 |
| 11 | 0.7857 | 11 | 0.2857 |
| 12 | 0.7143 | 12 | 0.0833 |
| 13 | 0.6222 | 13 | 0.2711 |
| 14 | 0.5871 | 14 | 0.2144 |
| 15 | 0.4286 | 15 | 0.1429 |
| Ave | 0.5966 | | 0.2284 |

Table 5.2: Number of participants, mean and estimated variance of each group.

| | Group 1 | Group 2 |
|---|---|---|
| Number of participants | $n_1 = 15$ | $n_2 = 15$ |
| Mean details recalled | $\bar{X}_1 = 0.5966$ | $\bar{X}_2 = 0.2284$ |
| Estimated variance | $s_1^2 = 0.0158$ | $s_1^2 = 0.0113$ |

## 5.4 Result Analysis

To analyze the data, we first calculate the means, and use Eqn (5.2) to get $s_1^2$ and $s_2^2$, each is an estimate of the population variance: $s_1^2$=0.0158, $s_2^2$=0.0113. The means and estimated variances of the two groups are listed in Table 5.2.

Next, we calculate pooled variance based on Eqn (5.3) and get $s_{pool}^2 = 0.0135$.

Then by using $s_{pool}^2$, based on Eqn (5.4), we calculate the standard error of the sampling distribution, called standard error of the difference, which is $s_{\bar{X}_1 - \bar{X}_2} = 0.0424$. Standard error of the difference estimates the difference between the two means.

Finally, based on Eqn (5.5), we calculate $t_{obt}$ and get $t_{obt} = 8.6690$.

In our experiment, $n_1$=15 and $n_2$=15, thus $df = (15 - 1) + (15 - 1) = 28$. With alpha at 0.05, the two-tailed $t_{crit}$ is 2.0484.

Figure 5.1 shows us the values on the sampling distribution of the difference. As $t_{obt} = +8.6690$, it is far beyond the non-reject region, so we can reject $H_0$ and accept $H_a$, which indicates that the difference between the two means is significant.

The experimental result suggested that by using the optimized teaching strategy, the rejection rate has been reduced from 0.5966 to 0.2284.

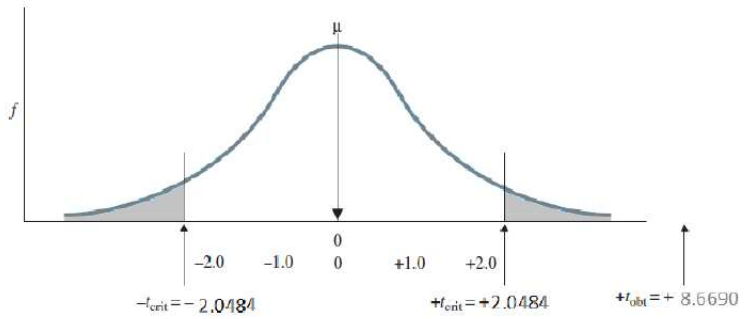


Figure 5.1: $H_0$ sampling distribution of differences between means when $\mu_1 - \mu_2 = 0$

# Chapter 6

# Conclusion

## 6.1 Major contributions

In responding student questions, an ITS should choose answers based on the student's study states, particularly what the student understands and what the student does not understand. However, in practical applications, an ITS does not always have the exact information about a student's study states. In this research, we develop a new technique on the basis of partially observable Markov decision process (POMDP), to address this issue. The technique enables an ITS to choose the best possible answers to student questions when information about a student's study states is partial and incomplete. The experiment has showed that the technique can remarkably improve an ITS's teaching performance.

In addition, we create a new architecture for building an ITS. In the architecture, states are defined in terms of the important concepts in the subject taught by the ITS. Defined in this way, the states are Markovian, and are suitable for COMDP and POMDP. Also, a state contains the most important information about the student. The system can obtain the information locally in choosing a right answer to a student question.

## 6.2 Future work

There is a big room for further improvement. Firstly, system parameters need to be adjusted for better performance, for example, the reward values. Secondly, in the current technique, there is only one agent. In the future work, we can extend the algorithm into a multi-agent reinforcement learning (MARL) algorithm. For example, we can develop a two-agent algorithm, in which there is a system agent and a user agent, the two agents share the same state space. The system agent is responsible for optimizing the teaching strategy and applying the strategy. The user agent models the user behavior and applies the information about the user behavior for disambiguating user input.

# Appendix A

# Example Concepts in Basic Level Software Tutoring

UML is an acronym for Unified Modeling Language, a widely accepted standard incorporating object orientation concepts.

Object orientation is a complete conceptual framework that covers the entire life-cycle of an IT project.

Objects are model elements that represent instances of a class or of classes.

The class diagram is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code.

A Link is the basic relationship among objects.

Composition represents wholepart relationships and is a form of aggregation.

Generalization is that any instance of the subtype is also an instance of the superclass.

An association specifies a semantic relationship that can occur between typed instances

Aggregation is an association that represents a part-whole or part-of relationship.

Composite aggregation is a strong form of aggregation that requires a part instance

be included in at most one composite at a time.

A realization relationship is a relationship between two model elements

Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time.

A multiplicity specification is a subset of the open set of non-negative integers.

Boundary classes handle the communication between actors and the system's internal components.

Entity classes model the information handled by the system.

Control classes handle the flow of control for a use-case and can therefore be seen as co-ordinating representation classes.

Polymorphism means the ability to take on many forms.

A use case is a use to which the system will be put that produces an observable result and usually provides value to one or more entities that interact with the system.

A business use case is an interaction with a business system.

A system use case is an interaction with an IT system.

Class is a user define data type.

Data type is generic to all programming paradigm.

Attributes are structural features of classes and data types.

Associations are structural relationships between classes that describe sets of links between objects.

Constraints are conditions or restrictions expressed in a natural or in a machine-readable language that extend the basic semantics of the model implied from the definition of the modeling language.

Queries provide a means to identify a piece of the entire object space of the running system according to certain matching criteria, and return a collection of tuples of objects and data values as a result.

Operations are behavioral features of classifiers that specify services that can be requested from the classifiers instances.

Methods can be specified in a detail level language that is supported by the implementation of the profile.

A state machine is a software engineering concept for modeling event-driven behavior of different kinds of entities.

Collaborations represent the vehicle for sketching, designing, illustrating, and documenting the structure of collaborating entities and their communication paths, along with the way they communicate in order to accomplish a certain common task.

Interactions are behavioral elements that describe or specify how participants in a collaboration communicate by exchanging messages.

A command is a class whose objects represent requests for a service from the system, issued interactively from the GUI or from outer systems interoperating with the given system.

A programming paradigm is a fundamental style of computer programming.There are four main paradigms: imperative, functional, object-oriented, and logic programming.

Imperative programming is a programming paradigm that describes computation in terms of statements that change a program state.

Object-oriented programming (OOP) is a programming paradigm that represents concepts as "objects" that have data fields (attributes that describe the object) and associated procedures known as methods.

Logic programming is one of the programming paradigms,its theory of computation is based on first order logic.

Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids state and mutable data.

Development tool is a program or application that software developers use to create, debug, maintain, or otherwise support other programs and applications.

Development environment refers to a server tier designated to a specific stage in a release process.

Local is a developer's desktop/workstation.

Virtual Machine is hosted on developers desktop or possibly development server.

Development server is a sandbox.

Integration,or continuous integration,builds target, or for developer testing of side effects

Test/QA is for functional, performance testing, Quality Assurance etc.

UAT stands for User acceptance testing.

Stage/Pre-production is the mirror of production environment.

Production/Live serves end-users/clients

Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware, thus making it behave as expected.

Anti-debugging is the implementation of one or more techniques within computer code that hinders attempts at reverse engineering or debugging a target process.

A debugger or debugging tool is a computer program that is used to test and debug other programs (the "target" program).

An algorithm is a step-by-step procedure for calculations in mathematics and computer science.

Computer algorithm is basically an instance of logic written in software by software developers to be effective for the intended "target" computer(s) for the target machines to produce output from given input (perhaps null).

A computer is a restricted type of machine, a discrete deterministic mechanical device that blindly follows its instructions.

Computer language is a simulation of an algorithm.

A programming language is an artificial language designed to communicate instructions to a machine, particularly a computer.

The syntax of a computer language is the set of rules that defines the combinations of symbols that are considered to be a correctly structured document or fragment in

that language.

Semantics refers to the meaning of languages.

The static semantics defines restrictions on the structure of valid texts that are hard or impossible to express in standard syntactic formalisms.

The dynamic semantics of a language defines how and when the various constructs of a language should produce a program behavior.

A type system defines how a programming language classifies values and expressions into types, how it can manipulate those types and how they interact.

A flowchart is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows.

A use case diagram provides a graphical overview of the functionality of a system.

An activity diagram is used to show a business or software process as a work flow through a series of actions.

A sequence diagram is used to show the sequence of interactions between classes, components, subsystems, or actors.

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes.

A logical class diagram shows that the member classifiers and packages are contained within the package.

A UML class diagram is to express the static relationship between classes, interfaces, and other components of a software system.

# Appendix B

# Concepts in the Example in the Text

A binary digit is 0 or 1.

A bit is the smallest unit of information on a computer. It can hold one of two values: 0 or 1.

A byte consists of eight consecutive bits.

Data represents information, stored on a computer as bits or bytes.

A computer file is a collection of data, which has a name.

An instruction is used to describe a rudimentary programming command.

In a machine language, each instruction is represented as binary digits.

An assembly language has the same structure and set of instructions as a machine language, with the instructions represented by names.

A high-level language is independent of any particular type of computer, and is closer to human languages than assembly languages.

A query language is a high-level language for requesting data.

A program is a sequence of instructions, for performing a specified task with a computer.

An application program is a program developed to performing a specific function

directly for the user.

A database is a collection of interrelated data files, along with a set of application programs written in a query language.

# Bibliography

[1] Ai, H. , Litman, D. J. , Forbes-Riley, K. , Rotaru, M. , Tetreault, J. , and Purandare, A. (2006). "Using system and user performance features to improve emotion detection in spoken tutoring dialogs". In *Proceedings of the International Conference on Spoken Language Processing (Interspeech 2006 (ICSLP)*, pages 797–800, Pittsburgh.

[2] Chickering, D. M. , and Paek, T. (2007). "Personalizing influence diagrams: applying online learning strategies to dialogue management". *User Model User-Adap Inter*, pp. 71-91, Springer, 2007.

[3] Chotimongkol, A. , and Rudnicky, A. I. (2001). "N-best Speech Hypotheses Reordering Using Linear Regression". *Proceedings of EuroSpeech 2001* (pp. 18291832), Aalborg, Denmark, September 3-7, 2001.

[4] Forbes-Riley, K. , Litman, D. , Huettner, A. , and Ward, A. (2005). "Dialogue-learning correlations in spoken dialogue tutoring". *Proceeding of the 2005 conference on Artificial Intelligence in Education: Supporting Learning through Intelligent and Socially Informed Technology*, 2005.

[5] Frampton, M. , and Lemon, O. (2006). "Learning more effective dialogue strategies using limited dialogue move features". *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 185–192, Sydney, Australia.

[6] Frampton, M. , and Lemon, O. (2009). "Recent research advances in reinforcement learning in spoken dialogue systems". *Knowledge Engineering Review*, 24(4):375–408, 2009.

[7] Gabsdil, M. , and Lemon, O. (2004). "Combining Acoustic and Pragmatic Features to Predict Recognition Performance in Spoken Dialogue Systems". *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics* (pp. 343-350), Barcelona, July 21-26, 2004.

[8] Gasic, M. ,Jurcicek, F. ,Thomson, B. , Yu, K. , and Young, S. (2011). "On-line policy optimisation of spoken dialogue systems via live interaction with human subjects". *Proceedings of IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU) 2011* (pp. 312-317), Hawaii, Dec 11-15, 2011.

[9] Gravano, A. , Hirschberg J. , and Benu, S. (2012). "Affirmative Cue Words in Task-Oriented Dialogue". *Computational Linguistics archive* Volume 38 Issue 1, March 2012 Pages 1-39.

[10] Hamid, R. C. ,Brahim, C. ,Luc, L. (2012). "Learning observation models for dialogue POMDPs". *Canadian AI'12 Proceedings of the 25th Canadian conference on Advances in Artificial Intelligence* , Pages 280-286, Springer-Verlag Berlin, Heidelberg 2012.

[11] Higashinaka, R. , Sudoh, K. , and Nakano, M. (2006). "Incorporating discourse features into confidence scoring of intention recognition results in spoken dialogue systems". *Speech Communication*, Vol 48(3-4): pp. 417-436, 2006.

[12] Iglesias, A. , Martnez, P. , and Fernndez, F. (2009). "Learning teaching strategies in an Adaptive and Intelligent Educational System through Reinforcement Learning". *Journal Applied Intelligence*, Volume 31 Issue 1, August 2009.

[13] Ivan, H. , and Miloslav, K. (2013). "SWSNL: Semantic Web Search Using Natural Language". *Expert Systems with Applications: An International Journal archive* Volume 40 Issue 9, July, 2013, Pages 3649-3664.

[14] Janarthanam, S. , Hastie, H. , Lemon, O. , and Liu, X. (2011). "The day after the day after tomorrow: a machine learning approach to adaptive temporal expression generation: training and evaluation with real users". *SIGDIAL '11 Proceedings of the SIGDIAL 2011 Conference* (Pages 142-151), Stroudsburg, PA, USA 2011.

[15] Jeremiah, T. F. , Gita, S. , and Sae, S. (2013). "Tractable POMDP representations for intelligent tutoring systems". *ACM Transactions on Intelligent Systems and Technology (TIST) - Special section on agent communication, trust in multi-agent systems, intelligent tutoring and coaching systems archive*, Volume 4 Issue 2, March 2013.

[16] Johnson, R. (2006). "Dialogue context-based re-ranking of ASR hypotheses". *Proceedings of IEEE 2006 Workshop on Spoken Language Technology* (174-177), Palm Beach, Aruba, Dec 10-13, 2006.

[17] Jurcicek, F. , Thomson, B. , Keizer, S. , Gasic, M. , Mairesse, F. , Yu, K. , and Young, S. (2010). "Natural Belief-Critic: a reinforcement algorithm for parameter estimation in statistical spoken dialogue systems". *Proceedings of Interspeech10* (pp 90-93), Sept 26-30, 2010.

[18] Lemon, O. , and Konstas, I. (2009). "User simulation for context-sensitive speech recognition in spoken dialogue systems". *Proceedings of the 12th Conference of the European Chapter of the ACL* (pp. 505-513), Athens, Greece, 30 March - 3 April, 2009.

[19] Levin, E. , Pieraccini, R. , and Eckert, W. (2000). "A stochastic model of human-machine interaction for learning dialogue strategies". *Speech and Audio Processing, IEEE Transactions on*, 8:11–23.

[20] Lidstone, G. J. (2010). "Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities". *Transactions of the Faculty of Actuaries*, Vol. 8, pp. 182-192, Aug 2010.

[21] Litman, A. J. , and Silliman, S. (2004). "Itspoke: an intelligent tutoring spoken dialogue system". In *Proceedings of Human Language Technology Conference 2004*.

[22] Pan, Y. ,Lee, H. , and Lee, L. (2012). "Interactive Spoken Document Retrieval With Suggested Key Terms Ranked by a Markov Decision Process". *IEEE Transactions on Audio, Speech, and Language Processing archive* Volume 20 Issue 2, February 2012, Page 632-645.

[23] Schatzmann, J. , Weilhammer, K. , Stuttle, M. , and Young, S. (2006). "A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies". *The Knowledge Engineering Review*, Vol. 21, 1-24. 2006.

[24] Scheffler, K. , and Young, S. (2002). "Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning". *Proceedings of the second international conference on Human Language Technology Research* (pp. 12-19). San Diego, California: Morgan Kaufmann Publishers Inc, 2002.

[25] Selfridge, E. O. , Arizmendi, I. ,Heeman, P. A. , and Williams J. D. (2012). "Integrating incremental speech recognition and POMDP-based dialogue systems". *SIGDIAL '12 Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue* Pages 275-279 Stroudsburg, PA, USA 2012.

[26] Sutton, R. S. , and Barto, A. G. (2005). "Reinforcement Learning: An Introduction". *The MIT Press*, Cambridge Massachusetts.

[27] Tetreault, J. , and Litman, D. (2006). "Using reinforcement learning to build a better model of dialogue state". *Proceedings of ECAL06*.

[28] Thomson, B. , and Young, S. (2010). "Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems". *Elsevier Computer Speech and Language*, Vol. 24, pp. 562-588, 2010.

[29] Thomson, B. , Jurcicek, F. , Gai, M. , Keizer, S. , Mairesse, F. , Yu, K. , and Young, S. (2010). "Parameter learning for POMDP spoken dialogue models". *Spoken Language Technology Workshop (SLT), 2010*, pp. 271-276, Berkeley, USA, 2010.

[30] Torres, F. , Sanchisk, E. , and Segarra, E. (2008). "User simulation in a stochastic dialog system". *Computer Speech and Language*, Vol 22, pp. 230-255, 2008.

[31] Wang, F. (2012). "Modeling User Behavior by Using a POMDP Reinforcement LearningAlgorithm". *Proceedings of IASTED Modeling and Simulation 2012 (MS2012), Banff, Canada, 3-5, July, 2012, pp. 304-310*, 2012.

[32] Wang, F. , and Kyle S. (2013). "Modeling user behavior online for disambiguating user input in a spoken dialogue system". *Speech Communication, Volume 55, Issue 1, January 2013 Pages 84-98.*

[33] Williams, J. D. (2011). "Integrating incremental speech recognition and POMDP-based dialogue systems". *SIGDIAL '11 Proceedings of the SIGDIAL 2011 Conference* Pages 130-141 Stroudsburg, PA, USA 2011.

[34] Williams, J. , Poupart, P. , and Young, S. (2005). "Factored Partially Observable Markov Decision Processes for Dialogue Management". *Proceedings of Knowledge and Reasoning in Practical Dialogue Systems*, 2005.

[35] Williams, J. D. , and Young, S. (2007). "Partially observable Markov decision processes for spoken dialog systems". *Elsevier Computer Speech and Language* Vol 21, pp. 393-422, 2007.