POMDP Framework for Building an Intelligent Tutoring System

Fangju Wang
School of Computer Science, University of Guelph, Guelph, Canada

Keywords: Intelligent Tutoring System, Partially Observable Markov Decision Process, Reinforcement Learning.

Abstract: When an intelligent tutoring system (ITS) teaches its human student on a turn-by-turn base, the teaching can be modeled by a Markov decision process (MDP), in which the agent chooses an action, for example, an answer to a student question, depending on the state it is in. Since states may not be completely observable in a teaching process, partially observable Markov decision process (POMDP) may offer a better technique for building ITSs. In our research, we create a POMDP framework for ITSs. In the framework, the agent chooses answers to student questions based on belief states when it is uncertain about the states. In this paper, we present the definition of physical states, reduction of a possibly exponential state space into a manageable size, modeling of a teaching strategy by agent policy, and application of the policy tree method for solving a POMDP. We also describe an experimental system, some initial experimental results, and result analysis.

1 INTRODUCTION

An intelligent tutoring system (ITS) is a computer system that teaches a subject to human students, usually in an interactive manner. An ITS may work on a platform of a regular desktop or laptop computer, a smaller device like a mobile phone, or the Internet. ITSs have advantages of flexibility in scheduling and pace, and so on. ITSs will play an important role in computer based education and training.

An ITS performs two major tasks when it teaches a student: interpreting student input (e.g. questions), and responding to the input. In this research, we address the problem of how to choose the most suitable response to a question, when the tutoring is conducted in a form of question-and-answer.

Many subjects (e.g. software basics, mathematics) can be considered to include a set of concepts. For example, the subject of "basic software knowledge" includes concepts of *binary digit*, *bit*, *byte*, *data*, *file*, *programming language*, *database*, and so on. Understanding the concepts is an important task in studying the subject, possibly followed by learning problem solving skills. In teaching such a subject, an ITS must teach the concepts. Quite often, a student studies a subject by asking questions about the concepts.

In a subject, concepts are interrelated. Among the relationships between concepts, an important one is the *prerequisite* relationship. Ideally, to answer a student question about a concept, an ITS should first teaches all the prerequisites of the concept that the student does not understand, and only those prerequisites. If the ITS talks about many prerequisites that the student already understands, the student may become impatient and the teaching would be inefficient. If the ITS misses a key prerequisite that the student does not understand, the student may become frustrated and the teaching would be ineffective.

To decide how to teach a student about a concept, it is essential for the ITS to determine the right set of prerequisites to "make up". The decision depends on the student's study *state*. It can be seen that the selection of right system responses can be modeled by a Markov decision process (MDP).

In practical applications, the study state of a student may not be completely observable to the system. That is, the system may be uncertain about the student's study state. To enable an ITS to make a decision when information about a state is uncertain, we apply the technique of *partially observable Markov decision process* (*POMDP*). We create a POMDP framework for building an ITS, and develop a *reinforcement learning* (*RL*) algorithm in the framework for choosing the most suitable answers to student questions.

The novelty of our work includes techniques for efficiently solving POMDP and dramatically reducing the state space. The great complexity in solving POMDP and exponential state space are two major issues to address in applying POMDP to building ITSs.

Wang F..
POMDP Framework for Building an Intelligent Tutoring System.
DOI: 10.5220/0004801702330240
In *Proceedings of the 6th International Conference on Computer Supported Education* (CSEDU-2014), pages 233-240
ISBN: 978-989-758-020-8
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

In this paper, we describe the POMDP framework, including the representation of student study states by POMDP states, representation of questions and answers by POMDP actions and observations, our technique for dealing with the exponential state space, modeling of teaching strategy by an agent policy, and the policy tree method for solving the POMDP.

2 RELATED WORK

POMDP had been applied in education in 1990s. In an early survey paper (Cassandra, 1998), the work for using POMDP to build teaching machines was reviewed, in which POMDP was applied to model internal mental states of individuals, and to find the best ways to teach concepts.

Recent work related with using RL and POMDP for tutoring dialogues include (Litman and Silliman, 2004), (William et-al, 2005), (Williams and Young, 2007), (Folsom-kovarik et-al, 2010), (Thomson et-al, 2010), (Rafferty et-al, 2011), (Chinaei et-al, 2012), and (Folsom-Kovarik et-al, 2013). In the following, we review in more details some representative work.

In the work (Theocharous et-al, 2009), researchers developed a framework called SPAIS (Socially and Physically Aware Interaction Systems), in which Social Variables defined the transition probabilities of a POMDP whose states are Physical Variables. Optimal teaching with SPAIS corresponded to solving an optimal policy in a very large factored POMDP.

The paper of (Rafferty et-al, 2011) presented a technique of faster teaching by POMDP planning. The researchers framed the problem of optimally selecting teaching actions using a decision-theoretic approach and showed how to formulate teaching as a POMDP planning problem. They considered three models of student learning and presented approximate methods for finding optimal teaching actions.

The work in (Folsom-kovarik et-al, 2010) and (Folsom-Kovarik et-al, 2013) studied two scalable POMDP state and observation representations. State queues allowed POMDPs to temporarily ignore less-relevant states, and observation chains represent information in independent dimensions.

The existing work of applying POMDP to build ITSs was characterized by off-line policy improvement. The costs of solving POMDP and searching in an exponential state space created great difficulties in building systems for practical teaching tasks. In our research, we aim to developing more efficient techniques for solving POMDP and reducing the state space into a manageable size, and to achive online policy improvement.

3 RL AND POMDP

3.1 Reinforcement Learning

Reinforcement learning (RL) is an interactive machine learning technique (Sutton and Barto, 2005). In an RL algorithm, there is one or a group of learning agents, which learn knowledge through interactions with the environment. A learning agent is also a problem-solver. It applies the knowledge it learns to solve problems. Meanwhile, it improves the knowledge in the process of problem-solving.

The major components of an RL algorithm are S, A, T and ρ, 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*, and ρ: $S \times A \times S \rightarrow \Re$ is the *reward function* where $\Re$ is a set of *rewards*.

At time step t, the agent is in state s_t, it takes action a_t. The action causes a state transition from s_t to a new state s_{t+1}. When the agent enters s_{t+1} at time step $t+1$, it receives reward $r_{t+1} = \rho(s_t, a_t, s_{t+1})$. The long term *return* at time step t is defined as

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

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

An additional component is *policy* π. π can be used to choose the optimal action to take in a state:

$$\pi(s) = \hat{a} = \arg\max_a Q^{\pi}(s,a), \tag{2}$$

where s is the state, $\hat{a}$ is the optimal action in s, and $Q^{\pi}(s,a)$ is the *action-value function* given π. It evaluates the expected return if a is taken in s and the subsequent actions are chosen by π:

$$Q^{\pi}(s,a) = \sum_{s'} P(s'|s,a) V^{\pi}(s') \tag{3}$$

where 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 π:

$$V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} P(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^{\pi}(s')] \tag{4}$$

where γ 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.

3.2 Partially Observable Markov Decision Process

The RL discussed above is based on Markov decision process (MDP), in which all the states are completely observable. For applications in which states are not completely observable, POMDP may provide a better technique (Kaelbling et al., 1998).

The major components of POMDP are S, A, T, ρ, O, Z, and b_0. The first four are the same as the counterparts in RL. 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 $o \in O$ after taking a and entering s'. b_0 is the *initial belief state*.

POMDP is differentiated from MDP by the introduction of *belief state* denoted by b:

$$b = [b(s_1), b(s_2), ..., b(s_N)] \tag{5}$$

where $s_i \in S$ ($1 \leq i \leq N$) is the ith state in S, N is the number of states in S, $b(s_i)$ is the probability that the agent is currently in s_i and $\sum_{i=1}^{N} b(s_i) = 1$. To avoid confusion, we refer to an $s \in S$ as a *physical state*.

At a point of time, the agent is in a physical state $s \in S$. Since states are not completely observable, the agent has only the probabilistic information about the states that it is in. The information is represented by b, as given in (5). Based on b, the agent chooses action a to take. After taking a, the agent enters $s' \in S$ and observes o. The process is showed in Figure 1. The total probability for the agent to observe o after a is

$$P(o|a) = \sum_{s\in S} b(s) \sum_{s'\in S} P(s'|s,a)P(o|a,s'). \tag{6}$$

Action a causes a physical state transition: The agent enters s', which is not observable either. The state information available to the agent is a new belief state b'. Each element in b' is calculated as

$$b'(s') = \sum_{s\in S} b(s)P(s'|s,a)P(o|a,s')/P(o|a) \tag{7}$$

where o is what the agent observes after taking a, and $P(o|a)$ defined in (6) is used as a normalization constant so that the elements in b' sum to one.

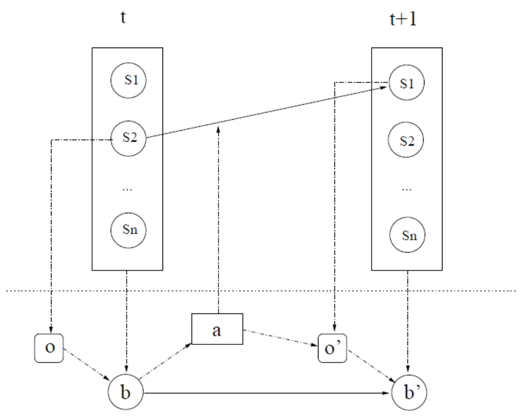

Figure 1: Physical states, belief states, actions, observations, and state transitions.

In POMDP, we use π to guide the agent to take actions. Differing from a policy in MDP, a policy in POMDP is a function of a belief state. The task to find the optimal policy is referred to as *solving a POMDP*.

The method of *policy tree* is used to simplify the process of solving a POMDP. In a policy tree, nodes represent actions, and edges represent observations. After the agent takes action a represented by a tree node, it observes o. The next action the agent will take is one of the children of a, connected by the edge representing o.

A policy tree is associated with a V function. In the following, we denote the V function of policy tree p as V_p. The value of physical state s given p is:

$$V_p(s) = \mathcal{R}(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \sum_{o\in O} P(o|a,s')V_{p(o)}(s') \tag{8}$$

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 γ is a reward discounting factor. The second term in the expression on the right side of (8) is the discounted expected value of future states.

The value of belief state b is

$$V_p(b) = \sum_{s\in S} b(s)V_p(s). \tag{9}$$

For belief state b, there is an optimal policy tree p, which maximizes the value of the belief state:

$$V(b) = \max_{p\in\mathcal{P}} V_p(b) \tag{10}$$

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

$$\pi(b) = \hat{p} = \arg\max_{p\in\mathcal{P}} V_p(b) \tag{11}$$

4 INTELLIGENT TUTORING SYSTEM ON POMDP

4.1 An Overview

We cast our ITS onto the framework of RL and POMDP. The main components of the ITS include states, actions, observations, and a policy. The states represent the student's study states: what the student understands, and what the student does not understand. The actions are the system's responses to student input, and the observations are student input, including questions. The policy represents the teaching strategy of the system.

At a point of time, the learning agent is in state s, which represents the agent's knowledge about the student's study state. Since in practical applications, the knowledge is not completely certain, we calculate belief state b for the knowledge. b is a function of the previous belief state, previous system action (e.g. answer), and the immediate student action (e.g. question) just observed by the agent. To respond to the student action, policy $\pi(b)$ is used to choose the most suitable system action a, for example, the answer to the student question.

After seeing system action a, the student may take another action, treated as observation o. Then new belief state b' is calculated from a and o, and the next system action is chosen by $\pi(b')$, and so on.

4.2 States: Student Study States

In teaching a student, the knowledge about the student's study state is essential to choose a teaching strategy. By *study state*, we mean what concepts the student understands and what concepts the student does not understand. We define the states in the POMDP to represent student study states.

As mentioned, a subject includes a set of concepts. For each concept C, we define two conditions:

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

We use expressions made of $\surd C$ and $\neg C$ to represent study states. For example, we can use expression $(\surd C_1 \surd C_2 \neg C_3)$ to represent that the student understands concepts C_1 and C_2, but not concept C_3.

A state is associated with an expression made of $\surd C$ and $\neg C$. We call such an expression a *state expression*. A state expression specifies the agent's knowledge about the student's study state. For example, when the agent is in a state associated with expression $(\surd C_1 \surd C_2 \neg C_3)$, the agent has the knowledge about the concepts that student understands and does not understands. When the subject taught has N concepts, each state expression is of the form

$$(\mathcal{C}_1\mathcal{C}_2\mathcal{C}_3...\mathcal{C}_N), \qquad (12)$$

where $\mathcal{C}_i$ takes $\surd C_i$ or $\neg C_i$ $(1 \leq i \leq N)$.

The major advantage of this state definition is that each state has the most important information required to teach the student – the study state. In addition, the states thus defined are Markovian. The information required for choosing a system response is available in the state that the agent is in.

4.3 Dealing with the Exponential State Space

As mentioned, when there are N concepts in the subject taught by the ITS, a state expression is of the form $(\mathcal{C}_1\mathcal{C}_2\mathcal{C}_3...\mathcal{C}_N)$, where $\mathcal{C}_i$ takes $\surd C_i$ or $\neg C_i$ $(1 \leq i \leq N)$. Thus 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 in which a student understands "byte" without knowledge of "bit", which is a prerequisite of "byte".

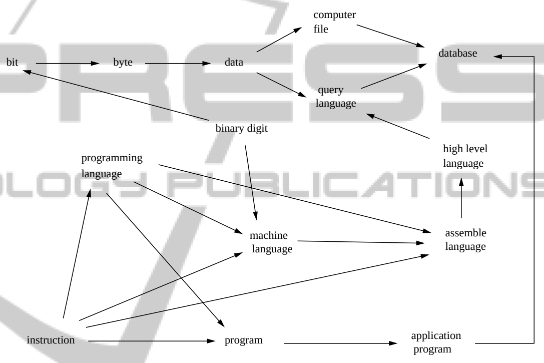

Figure 2: A DAG showing prerequisites between concepts.

To deal with the exponential space, we use the relationship of concept prerequisites in encoding state expressions. The relationship helps eliminate invalid states and maintain a state space of a manageable size. Let C_1 and C_2 be two concepts in a subject. If to understand concept C_2, a student must first understand C_1, we say C_1 is a prerequisite of C_2. A concept may have one or more prerequisites, and a concept may serve as a prerequisite of one or many other concepts. The concepts and their prerequisite relationships can be represented in a directed acyclic graph (DAG). Figure 2 is such a DAG for a subset of concepts in basic software knowledge.

When encoding state expressions in the form of (12), we perform a topological sorting on the DAG, to generate a 1-D sequence of the concepts. For example, the following is a topologically sorted sequence of the concepts in the DAG in Figure 2:

BD BI BY DA FI IN PL ML AL HL QL PR AP DB

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".

In a topologically sorted state expression, all the direct and indirect prerequisites of a concept are on the left hand side of it. The sorting helps determine 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. In a state expression, if the jth concept is in $\surd C_j$ condition, and there exists a prerequisite left to it, e.g. the ith concept, which in condition $\neg C_i$, we can determine that the state is invalid by using simple calculation.

Let C_{j1}, C_{j2}, ... C_{jN_j} be the set of prerequisites of C_j. We call $C_{j1}C_{j2}...C_{jN_j}C_j$ the *prerequisite sequence* of C_j. A prerequisite sequence is in a valid condition, if and only if when concept C_k in the sequence is $\surd C_k$, any concept C_i to its left is $\surd C_i$. A state expression made of N concepts has at most N prerequisite sequences, and each of the prerequisite sequences has at most N valid conditions. We can thus estimate that the maximum number of valid states is N^2.

4.4 Definition of 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.

We classify actions in the ITS into *student actions* and *system actions*. Student actions mainly includes the actions of asking questions about concepts. Asking "what is a database?" is such an action. Each student action involves only one concept. In the following discussion, we denote a student action of asking about concept C by $[C]$.

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 in a query language". We use $\{C\}$ to denote a system action of explaining C.

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 explains C_j and C_k in order to eventually explain C_l. In the following discussion, we refer to such a sequence of actions for answering a question as a *answer path*.

5 OPTIMIZATION OF TEACHING STRATEGY

5.1 Teaching Strategy as Policy Trees

As discussed, 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 paper, *teaching strategy* is used to select of the starting concept in answering a question. The teaching strategy largely determines student satisfaction and teaching efficiency.

A 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 a prerequisite. The second possible action is to start with C_1, then teach C_2, until C_k. The third action 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. A disadvantage is the student may become frustrated and has to ask about many prerequisites. Another answer is to start with a very basic prerequisite. A disadvantage of this answer is low efficiency and the student may become impatient. When answering a student question, the system should choose a answer which starts with the "right" prerequisite if the concept in question has prerequisites.

We model the teaching strategy as the policy. 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 observation, 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 (11).

For each concept, we create a set of policy trees to answer 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 policy tree p that contain the most suitable answer path, based on the student's current study state.

5.2 Policy Initialization

From (11), (9), (10), and (8), 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. We define the reward function as $S \times A \rightarrow \Re$, and have $\mathcal{R}(s,a) = \rho(s,a)$, which returns rewards depending on if action a taken in state s is accepted or rejected by the student. More about rejection and acceptance of a system action will be given in the discussion of experimental results.

Policy initialization mainly involves creating $P(s'|s,a)$ and $P(o|a,s')$. We will see that updating them is also the primary task in policy improvement.

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

$$P(s'|s,a) = P(s_{t+1} = s'|s_t = s, a_t = a) \quad (13)$$

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

$$P(o|a,s') = P(o_{t+1} = o|a_t = a, s_{t+1} = s') \quad (14)$$

where t and $t+1$ are the same as (13).

To initialize $P(s'|s,a)$ and $P(o|a,s')$, we create action sequences as training data. The data are represented as tuples:

$$...(\breve{s}_t, a_t, o_t, \breve{s}_{t+1})(\breve{s}_{t+1}, a_{t+1}, o_{t+1}, \breve{s}_{t+2})... \quad (15)$$

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

$$P(s'|s,a) = \frac{|\text{transition from } s \text{ to } s' \text{ when } a \text{ is taken in } s|}{|a \text{ is taken in } s|}, \quad (16)$$

$$P(o|a,s') = \frac{|a \text{ is taken, } s' \text{ is perceived, } o \text{ is observed}|}{|a \text{ is taken and } s' \text{ is perceived}|}, \quad (17)$$

where $|\ |$ is the operator for counting. In (16) and (17) the counts are from the training data in the form of (15).

5.3 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 improves its policy through the interaction with the environment, and the policy improvement is conducted when the agent applies the policy to solve problems.

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 sessions, system and student actions are recorded. 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

$$...(\hat{s}_t, a_t, o_t, \hat{s}_{t+1})(\hat{s}_{t+1}, a_{t+1}, o_{t+1}, \hat{s}_{t+2})... \quad (18)$$

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

$$...(\breve{s}_t, a_t, o_t, \breve{s}_{t+1})(\breve{s}_{t+1}, a_{t+1}, o_{t+1}, \breve{s}_{t+2})... \quad (19)$$

From (19), we derive sequence

$$...(\breve{s}_t, a_t, \breve{s}_{t+1})(\breve{s}_{t+1}, a_{t+1}, \breve{s}_{t+2})... \quad (20)$$

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

$$...(a_t, o_t, \check{s}_{t+1})(a_{t+1}, o_{t+1}, \check{s}_{t+2})... \quad (21)$$

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

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

$$P(s'|s,a) = C_1/(C_2+C_4) + C_3/(C_2+C_4) \quad (22)$$

where

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

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

6 EXPERIMENTS AND RESULTS

6.1 Experimental System

An experimental system has been developed for implementing the techniques. It is an interactive ITS for teaching basic software knowledge. It teaches software knowledge in terms of about 150 concepts. The system teaches a student at a time on a turn-by-turn basis: The student asks a question about a concept, and the system answers the question, then based on the system answer the student asks a new question or asks a question for understanding the concept just questioned, and the system answers, and so on. The student communicates with the system by using a keyboard, and the system's output device is the screen. The ITS is a part of a larger project of a spoken dialogue system (SDS). Using the keyboard and screen for input and output allows us to focus on the development and improvement of the teaching strategy, without considering issues in speech recognition.

The main components of the system include a student module, an agent module, and a collection of databases. The student module is responsible for interpreting student input and converting it into a form usable to the agent module. (The student module and the input interpretation function are not discussed in this paper because of the limitation of paper length.) The agent module is the dialogue manager. For a student input (mostly a question), the agent module invokes policy $\pi(b)$ to choose the most suitable response.

The databases are the *system action database* storing human understandable system responses (answers), *policy tree database* storing policy trees for solving POMDP, *transition probability database* storing $P(s'|s,a)$, *observation probability database* storing $P(o|a,s')$, and *reward database* storing $\mathcal{R}(s,a)$.

6.2 Experiment

30 people participated in the experiment. In the following, we call them students. The students know how to use desktop or laptop computers, Windows or Mac operating systems, and application programs like Web browsers, word processors, and so on. They did not have formal training in computer science and software development.

The 30 students were randomly divided into two groups of the same size. The students in Group 1 studied with the ITS which did not have the improved teaching strategy. When a student asked about a concept, the system either explained the concept directly, or randomly chose a prerequisite to start. The students in Group 2 studied with the ITS in which the teaching strategy was continuously improved.

The ITS taught a student at a time. Each student studied with the ITS for about 45 minutes. For each student, the question-answer sessions were recorded for performance analysis.

The performance perimeter is *rejection rate*. Roughly, if right after the system explains concept C, the student asks a question about a prerequisite of C, or says "I already know C", we consider the student rejects the system action. For a student session, the rejection rate is defined as the ratio of the number of system actions rejected by the student to the total number of system actions.

6.3 Result Analysis

We applied a two-sample t-test method to evaluate the effects of the optimized teaching strategy to the teaching performance of an ITS. The test method is the *independent-samples t-test* (Heiman, 2011).

For each student, we calculated the mean rejection rate. For the two groups, we calculated means $\bar{X}_1$ and $\bar{X}_2$. Sample mean $\bar{X}_1$ is used to represent population mean μ_1, and $\bar{X}_2$ represent μ_2.

The alternative and null hypotheses are:

$$H_a : \mu_1 - \mu_2 \neq 0, \qquad H_0 : \mu_1 - \mu_2 = 0$$

The means and variances calculated for the two groups are listed in Table 1. In the experiment, n_1=15

Table 1: Number of students, mean and estimated variance of each group.

	Group 1	Group 2
Number of students	$n_1 = 15$	$n_2 = 15$
Sample mean	$\bar{X}_1 = 0.5966$	$\bar{X}_2 = 0.2284$
Estimated variance	$s_1^2 = 0.0158$	$s_1^2 = 0.0113$

and n_2=15, thus the degree of freedom is $(15-1)+(15-1) = 28$. With alpha at 0.05, the two-tailed t_{crit} is 2.0484 and we calculated $t_{obt} = +8.6690$. Since the t_{obt} is far beyond the non-reject region defined by $t_{crit} = 2.0484$, we should reject H_0 and accept H_a.

As listed in Table 1, the mean rejection rate in Group 1 was 0.5966 and the mean rejection rate in Group 2 was 0.2284, and the accepted alternative hypothesis indicated the difference between the two means was significant. The analysis suggested that by using the optimized teaching strategy, the rejection rate has been reduced from 0.5966 to 0.2284.

7 CONCLUSIONS

In teaching a student, an effective teacher should be able to adapt a suitable teaching strategy based on his/her knowledge about the student's study state, and should be able to improve his/her teaching when becoming more experienced. An effective ITS should have the same abilities. In our research, we attempt to build such an ITS. Our approch is POMDP.

Our research has novelty in state definition, POMDP solving, and online strategy improvement. The state definition allows important information to be available locally for choosing the best responses, and reduces an exponential space into a polynomial one. Compared with the existing work for applying RL and POMDP to build ITSs, which mainly depend on off-line policy improvement, our online improvement algorithm enables the system to continuously optimize its teaching steategies while it teaches.

ACKNOWLEDGEMENTS

This research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

Cassandra, A. (1998) A survey of pomdp applications. In *Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Process*, 17-24.

Chinaei, H. R., Chaib-draa, B., Luc Lamontagne, L. (2012). Learning observation models for dialogue POMDPs. *Canadian AI'12 Proceedings of the 25th Canadian conference on Advances in Artificial Intelligence* , 280-286, Springer-Verlag Berlin, Heidelberg.

Folsom-Kovarik, J. T., Sukthankar, G., Schatz, S., and Nicholson, D. (2010) Scalable POMDPs for Diagnosis and Planning in Intelligent Tutoring Systems. In *Proceesings of AAAI Fall Symposium on Proactive Assistant Agents 2010*.

Folsom-Kovarik, J. T., Sukthankar, G., and Schatz, S. (2013). Tractable POMDP representations for intelligent tutoring systems. *ACM Transactions on Intelligent Systems and Technology (TIST) - Special section on agent communication, trust in multiagent systems, intelligent tutoring and coaching systems archive*, 4(2), 29.

Heiman, G. W. (2011). *Basic Statistics for the Behavioral Sciences, Sixth Edition*. Wadsworth Cengage Learning, Belmont, California, USA.

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*, 90-93, Sept 26-30.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Elsevier Artificial Intelligence*, 101: 99–134.

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

Rafferty, A. N., Brunskill, E., Thomas L. Griffiths, T. L., and Patrick Shafto, P., (2011). Faster Teaching by POMDP Planning. In *Proceesings of Artificial Intelligence in Education (AIED) 2011)*, 280-287.

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

Theocharous, G., Beckwith, R., Butko, N., and Philipose, M. (2009). Tractable POMDP planning algorithms for optimal teaching in SPAIS. In *IJCAI PAIR Workshop (2009)*.

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*, 271-276, Berkeley, USA.

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

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