A New Technique of Policy Trees for Building a POMDP based Intelligent Tutoring System

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

Keywords: Intelligent Tutoring System, Partially Observable Markov Decision Process, Pomdp Solving, Policy Tree.

Abstract: Partially observable Markov decision process (POMDP) is a useful technique for building intelligent tutoring systems (ITSs). It enables an ITS to choose optimal tutoring actions when uncertainty exists. An obstacle to applying POMDP to ITSs is the great computational complexity in decision making. The technique of policy trees may improve the efficiency. However, the number of policy trees is normally exponential, and the cost for evaluating a tree is also exponential. The technique is still too expensive when applied to a practical problem. In our research, we develop a new technique of policy trees for better efficiency. The technique is aimed at minimizing the number of policy trees to evaluate in making a decision, and reducing the costs for evaluating individual trees. The technique is based on pedagogical orders of the contents in the instructional subject. In this paper, we first provide the background of ITS and POMDP, then describe the architecture of our POMDP based ITS, and then present our technique of policy trees for POMDP solving, and finally discuss some experimental results.

1 INTRODUCTION

Intelligent tutoring systems (ITSs) have become useful teaching aids in computer supported education. An ITS is a computer system that performs one-to-one, interactive, adaptive tutoring. Research in education science discovered that one-to-one interactive tutoring can achieve better teaching results than linear classroom lecturing (Bloom, 1984). ITSs are developed to offer the benefit of one-to-one interactive tutoring without the costs of dedicating one human teacher to each student. In recent years, ITSs have been applied in many fields including mathematics (Woolf, 2009), physics (Vanlehn, 2010), medical science (Woolf, 2009), and web-based adult education (Cheung, 2003).

Adaptive teaching is a key feature of an ITS. In each tutoring step, it chooses the optimal action according to the current knowledge state and affective state of its student. A knowledge state is a representation of the student's mastery of the instructional subject, while an affective state is an indicator of the student's frustration, boredness, etc.

Information about students' current states plays an important role in adaptive tutoring. However, in a practical tutoring process, the student's states may not be completely observable to the teacher. Quite often, the teacher does not know exactly what the student's states are, and what the most beneficial tutoring actions should be (Woolf, 2009). For building an adaptive tutoring system, partially observable Markov decision process (POMDP) provides useful tools to deal with the uncertainties. It enables a system to take optimal actions even when information of states is uncertain and/or incomplete.

In a POMDP, the information about student states is modeled by a set of states. Note we use POMDP states to model student states. At a point of time, the decision agent is in a state, which represents the current state of the student. The agent chooses the most beneficial action based on what the current state is.

Finding the optimal solutions (or actions) is the task of *POMDP solving*. A practical technique is to use *policy trees*, in which decision making involves evaluating policy trees and choosing the optimal one. The technique of policy trees is still very expensive, although it is better than many others for POMDP solving. In making a decision, the number of policy trees to evaluate is normally exponential in the number of possible observations and POMDP horizon (Carlin and Zilberstein, 2008). The cost for evaluating a policy tree is also exponential in the two variables (Rafferty et-al, 2011). The computational complexity discourages the application of the policy tree

Wang, F.
A New Technique of Policy Trees for Building a POMDP based Intelligent Tutoring System.
In *Proceedings of the 8th International Conference on Computer Supported Education (CSEDU 2016) - Volume 1*, pages 85-93
ISBN: 978-989-758-179-3

technique to practical tutoring problems.

We develop a novel technique of policy trees, aiming at minimizing the number of trees to evaluate in making a decision, and reducing the costs for evaluating individual trees. This technique is based on the information of pedagogical order of the contents in the instructional subject. In this paper, we first provide the background knowledge of ITS and POMDP, review some existing work of using POMDP for building ITSs, with emphasis on POMDP solving, then we present our technique of policy trees, and finally we discuss some experimental results.

2 INTELLIGENT TUTORING SYSTEMS

Two major features of an ITS are knowledge tracking and adaptive instruction: An ITS should be able to store and track a student's knowledge states during a tutoring process, and choose the optimal tutoring actions accordingly.

The core modules in an ITS include a domain model, a student model, and a tutoring model. The *domain model* stores the domain knowledge, which is basically the knowledge in the instructional subject. For a subject, an ITS may teach concepts or problem-solving skills, or both. In the domain model, the knowledge for the former is usually declarative, while the knowledge for the latter is procedural.

The *student model* contains information about students. There are two types of student information: the information about the behavior of general students in studying the subject, and the information about the current state of the student being tutored. The *tutoring model* represents the system's tutoring strategies.

In each tutoring step, the agent accesses the student model to obtain information about the student's current state, then based on the information it applies the tutoring model to choose a tutoring action and retrieves the domain model for the knowledge to teach. After taking the action, it updates the student model, chooses and takes the next action based on the updated model, and so on, till the tutoring session ends.

The above discussion suggests that intelligent tutoring can be modeled by a Markov decision process (MDP). In MDP, the decision agent is in a state at any point of time. Based on information of the state, it chooses and takes the action it considers optimal. After the action, the agent receives an award and enters a new state, where it chooses the next action, and so on. In MDP, states are completely observable to the decision agent, and the agent knows exactly what the current state is. However, as mentioned before, in a tutoring process a student's states are not always completely observable. Partially observable Markov decision process (POMDP) is a more suitable modeling tool for intelligent tutoring processes.

3 PARTIALLY OBSERVABLE MARKOV DECISION PROCESS

The major components of a POMDP are S, A, T, ρ, O, and Z, where S is a set of states, A is a set of actions, T is a set of state transition probabilities, ρ is a reward function, O is a set of observations, and Z is a set of observation probabilities. At a point of time, the decision agent is in state $s \in S$, it takes action $a \in A$, then enters state $s' \in S$, observes $o \in O$, and receives award $r = \rho(s,a,s')$. The probability of transition form s to s' after a is $P(s'|s,a) \in T$. The probability of observing o in s' after a is $P(o|a,s') \in Z$. Since the states are not completely observable, the agent infers state information from its observations, and makes decisions based on its inferred *beliefs* about the states.

An additional major component in POMDP is the *policy* denoted by π. It is used by the agent to choose an action based on its current belief:

$$a = \pi(b) \tag{1}$$

where b is the belief, which is defined as

$$b = [b(s_1), b(s_2), ..., b(s_Q)] \tag{2}$$

where $s_i \in S$ ($1 \leq i \leq Q$) is the ith state in S, Q is the number of states in S, $b(s_i)$ is the probability that the agent is in s_i, and $\sum_{i=1}^{Q} b(s_i) = 1$.

Given a belief b, an optimal π returns an optimal action. For a POMDP, finding the optimal π is called *solving the POMDP*. For most applications, solving a POMDP is a task of great computational complexity. A practical method for POMDP-solving is using *policy trees*. In a policy tree, nodes are actions and edges are observations. Based on a policy tree, after an action (at a node) is taken, the next action is determined by what is observed (at an edge). Thus a path in a policy tree is a sequence of "action, observation, action, observation, ..., action".

In the method of policy trees, a decision is to choose the optimal policy tree and take the root action. Each policy tree is associated with a value function. Let τ be a policy tree and s be a state. The value function of s given τ is

$$V^{\tau}(s) = \mathcal{R}(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \sum_{o \in O} P(o|a,s') V^{\tau(o)}(s') \tag{3}$$

where a is the root action of τ, γ is a discounting factor, o is the observation after the agent takes a, $\tau(o)$ is

the subtree in τ which is connected to the node of a by the edge of o, and $\mathcal{R}(s,a)$ is the expected immediate reward after a is taken in s, calculated as

$$\mathcal{R}(s,a) = \sum_{s' \in S} P(s'|s,a)\mathcal{R}(s,a,s') \quad (4)$$

where $\mathcal{R}(s,a,s')$ is the expected immediate reward after the agent takes a in s and enters s'. The second term on the right hand side of Eqn (3) is the discounted expected value of future states.

From Eqns (2) and (3), we have the value function of belief b given τ:

$$V^{\tau}(b) = \sum_{s \in S} b(s)V^{\tau}(s). \quad (5)$$

Then we have $\pi(b)$ returning the optimal policy tree $\hat{\tau}$ for b:

$$\pi(b) = \hat{\tau} = \arg\max_{\tau \in \mathcal{T}} V^{\tau}(b), \quad (6)$$

where $\mathcal{T}$ is the set of policy trees to evaluate in making the decision.

Now we discuss how the belief is updated. After the agent takes a in s, it enters into s' and observes o. It then has a new belief b'. the new belief is a function of b, a, o, and s'. Recall a belief is a vector (see Eqn (2)). Updating a belief is done by calculating the individual elements. The following is the formula actually used to calculate element $b'(s')$ in b':

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

where $P(o|a)$ is the total probability for the agent to observe o after taking a. It is calculated as

$$P(o|a) = \sum_{s \in S} b(s) \sum_{s' \in S} P(s'|s,a)P(o|a,s'). \quad (8)$$

$P(o|a)$ is used in Eqn (7) as a normalization factor so that the elements in b' sum to one.

From the above description, we can see that each decision step, which consists of choosing an action (by using Eqns (3), (4), (5), and (6)) and updating the belief (by using Eqns (7) and (8)), requires computation over the entire state space S and solution space $\mathcal{T}$. The two exponential spaces have been a bottleneck in applying POMDP to practical problems.

4 RELATED WORK

The work of applying POMDP to computer supported education started in as early as 1990s (Cassandra, 1998). In the early years' work, POMDP was used to model internal mental states of individuals, and to find the best ways to teach concepts. Typically, the states of a student had a boolean attribute for each of the concepts, the actions available to the teacher were various types of teaching techniques, and the observations were the results of tests given periodically. The goal could be to teach as many of the concepts in a finite amount of time, or to minimize the time required to learn all the concepts.

The recent work related with applying POMDP to intelligent tutoring included (Williams et al, 2005), (Williams and Young, 2007), (Theocharous et-al, 2009), (Rafferty et-al, 2011), (Chinaei et-al, 2012), and (Folsom-Kovarik et-al, 2013). The work was commonly characterized by using POMDP to optimize and customize teaching, but varied in the definitions of states, actions, and observations, and in the strategies of POMDP-solving. In the following, we review some representative work in more details, with emphasis on POMDP-solving.

In the work reported in (Rafferty et-al, 2011), the researchers created a system for concept learning. They developed a technique of faster teaching by POMDP planning. The technique was for computing approximate POMDP policies, which selected actions to minimize the expected time for the learner to understand concepts. The researchers framed the problem of optimally selecting teaching actions by using a decision-theoretic approach, and formulated teaching as a POMDP planning problem. In the POMDP, the states represented the learners' knowledge, the transitions modeled how teaching actions stochastically changed the learners' knowledge, and the observations indicated the probability that a learner would give a particular response to a tutorial action. Three learner models were considered in defining the state space: memoryless model, discrete model with memory, and continuous model.

For solving the POMDP, the researchers developed an online method of forward trees, which are variations of policy trees. A forward tree is constructed by interleaving branching on actions and observations. For the current belief, a forward trees was constructed to estimate the value of each pedagogical action, and the best action was chosen. The learner's response, plus the action chosen, was used to update the belief. And then a new forward search tree was constructed for selecting a new action for the updated belief. The cost of searching the full tree is exponential in the task horizon, and requires an $O(|S|^2)$ operations at each node. To reduce the number of nodes to search through, the researchers restricted the tree by sampling only a few actions. Additionally, they limited the horizon to control the depth of the tree.

The work described in (Folsom-Kovarik et-al, 2013) was aimed at making POMDP solvers feasible for real-world problems. The researchers created

a data structure to describe the current mental status of a particular student. The status was made up of knowledge states and cognitive states. The knowledge states were defined in terms of gaps, which are misconceptions regarding the concepts in the instructional domain. Observations are indicators that particular gaps are present or absent. The intelligent tutor takes actions to discover and remove all gaps. The cognitive states tracked boredom, confusion, frustration, etc. The intelligent tutor accounts for a learner's cognitive state so as to remove gaps more effectively.

To facilitate POMDP solving, the researchers developed two scalable representations of POMDP states and observation: state queue and observation chain. They introduced parameter d_{jk} for describing the difficulty of tutoring concept j before concept k. By reordering the gaps to minimize the values in d, a strict total ordering over the knowledge states, or priority, can be created. A state queue only maintained a belief about the presence or absence of one gap, the one with the highest priority. The state queues allowed a POMDP to temporarily ignore less-relevant states. The state space in a POMDP using a state queue was linear, not exponential.

The existing techniques for improving POMDP solving have made good progress towards building practical POMDP based ITSs. However they had limitations. For example, as the authors of (Rafferty et-al, 2011) concluded, computational challenges still existed in their technique of forward trees, despite sampling only a fraction of possible actions and using very short horizons. Also, how to sample the possible actions and how to shorten the horizon are challenging problems. As the authors of (Folsom-Kovarik et-al, 2013) indicated, the methods of state queue and observation chain might cause information loss, which might in turn degrade system performance in choosing optimal actions.

5 AN ARCHITECTURE OF POMDP ITS

We develop an experimental system as a test bed for our techniques, including the technique for POMDP solving in ITSs. In this section, we discuss how we cast an ITS onto the POMDP, and how we define states, actions, and observations.

5.1 Casting an ITS onto POMDP

The instructional subject of the ITS is the basic knowledge of software. The system is for concept learning. It tutors a student at a time, in a turn-by-turn interactive way. In a tutoring session, the student asks questions about software concepts, and the system chooses the optimal tutoring actions based on its information about the student's current states. POMDP is used for choosing the optimal tutoring actions.

Most concepts in the subject have prerequisites. When the student asks about a concept, the system determines whether it should start with explaining a prerequisite for the student to make up some required knowledge, and, if so, which one to explain. The optimal action is to explain the concept that the student needs to make up in order to understand the originally asked concept, and that the student can understand it without making up any other concepts.

We cast the ITS student model onto the POMDP states, and represent the tutoring model as the POMDP policy. At the current stage, the student model contains information about knowledge states only. In the architecture, ITS actions are represented by POMDP actions, while student actions are treated as POMDP observations.

At any point in a tutoring process, the decision agent is in a POMDP state, which represents the agent's information about the student's current state. Since the states are not completely observable, the agent infers the information from its immediate action and observation (the student action), and represents the information by the current belief. Based on the belief, the agent uses the policy to choose the optimal action to respond to the student.

5.2 Defining States

We define the states in terms of the concepts in the subject. In software basics, the concepts are *data*, *program*, *algorithm*, and many others. We use a boolean variable to represent each concept: Variable C_i represents concept C_i. C_i may take two values $\surd C_i$ and $\neg C_i$. $\surd C_i$ indicates that the student understands concept C_i, while $\neg C_i$ indicates that the student does not.

A conjunctive formula of such values may represent information about a student knowledge state. For example, $(\surd C_1 \wedge \surd C_2 \wedge \neg C_3)$ represents that the student understands C_1 and C_2, but not C_3. When there are N concepts in a subject, we can use formulas of N variables to represent student knowledge states. For simplicity, we omit the $\wedge$ operator, and thus have formulas of the form:

$$(C_1 C_2 C_3 ... C_N) \tag{9}$$

where C_i may take $\surd C_i$ or $\neg C_i$ $(1 \leq i \leq N)$. We call a formula of (9) a *state formula*. It is a representation of which concepts the student understands and which concepts the students does not.

In the POMDP, each state in S is associated with a state formula. When the decision agent is in a state, from the formula of the state, it has the information of the student's current knowledge state.

5.3 Actions and Observations

In a tutoring session, asking and answering questions are the primary actions of the student and system. Other actions are those for greeting, confirmation, etc.

In an ITS for concept learning, student actions are mainly asking questions about concepts. Asking "what is a query language?" is such an action. We assume that a student action concerns only one concept. In this paper, we denote a student action of asking about concept C by $(?C)$, and use (Θ) to denote an *acceptance* action, which indicates that the student is satisfied by a system answer, like "I see", "Yes", "please continue" or "I am done".

The system actions are mainly answering questions about concepts, like "A query language is a high-level language for querying." We use $(!C)$ to denote a system action of explaining C, and use (Φ) to denote a system action that does not explain a concept, for example a greeting.

6 THE NEW TECHNIQUE OF POLICY TREES

The goal of our technique is to minimize the number of trees to evaluate when the agent makes a decision, and to reduce the costs for evaluating individual trees. The technique is based on the information of prerequisite relationships.

6.1 Prerequisite Relationships in Learning

In most science subjects, there are pedagogical orders for teaching/learning contents. The relationship of content prerequisites is a pedagogical order. If, to well understand concept C_j, a student must first understand concept C_i, we call C_i a *prerequisite* of C_j. For example, in the subject of software basics, *data* is a prerequisite of *database*, and in mathematics, *function* is a prerequisite of *derivative*. A concept may have zero, one or more prerequisites, and a concept may be a prerequisite of zero, one or more other concepts. In this paper, when C_i is a prerequisite of C_j, we call C_j a *successor* of C_i. Figure 1 illustrate a directed acyclic graph (DAG) representing the direct prerequisite relationships between a subset of the concepts in software basics. Note that the relationships in the graph may not be complete and accurate. They are used in examples for discussing our technique only.

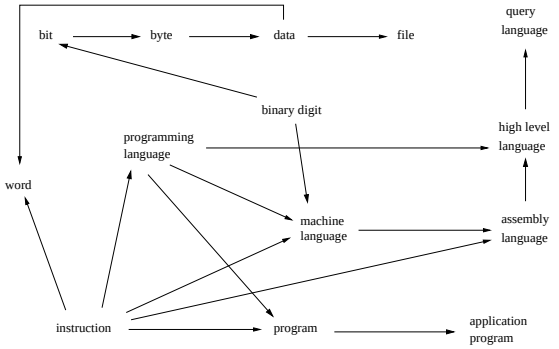

Figure 1: The DAG representing direct prerequisite relationships in a subset of the concepts in software basics.

In our research, we observed, through examining real tutoring sessions between human students and teachers, that the concepts asked by a student in a tutoring session usually have prerequisite/successor relationships with each other. Quite often, right after asking about a concept, a student may ask about a prerequisite of the concept. This happens when the system's answer makes the student to realize that he/she needs to make up some knowledge. For example, the student starts a tutoring session with question "What is a database?" After the system explains "database" in terms of "file", the student may ask "What is a file?". Sometimes, after a system's answer about a concept, a student may ask about a related concept, like a successor of the concept originally asked. This usually happens when the student has been satisfied with the answer and wants to learn more. For example, the student may ask a question about "data warehouse" after being satisfied by an explanation about "database".

Based on the observation, we develop a set of techniques in which the information of prerequisites is used to group concepts into subspaces, and to reduce the costs in evaluating policy trees. In the following, we discuss our algorithms for tree construction and application.

6.2 Policy Trees and Atomic Trees

We classify student questions into the *original questions* and *current questions*. An original question starts a tutoring session, and a current question is to be answered in the current step. In the above example of "database" and "file", "What is a database?" is the original question, and "what is a file?" is the current question before the system answers it.

The concept in the original question is the one that the student originally wants to learn. The student asks the current question usually for understanding the original question. Sometimes, the current question may also be made by the agent for the student to make up some knowledge. At the beginning of a session, the original question is also the current question.

In our discussion, we denote the original and current questions by $(?C^o)$ and $(?C^u)$ respectively, and assume $C^u \in (\wp_{C^o} \cup C^o)$, where $\wp_{C^o}$ is the set of all the direct and indirect prerequisites of C^o. For the pair of $(?C^o)$ and $(?C^u)$, we construct a set of policy trees. We denote the set by $\mathcal{T}_{C^u}^{C^o}$. *When the original question is* $(?C^o)$ *and current question is* $(?C^u)$*, to answer* $(?C^u)$ *the agent evaluates all the policy trees in* $\mathcal{T}_{C^u}^{C^o}$*, selects the optimal, and takes the root action of it.* The agent evaluates a policy tree to estimate the expected return that results from taking the root action in the current step.

We construct a policy tree by integrating some *atomic trees*. In this subsection, we discuss how to create atomic trees, and in the next subsection, how to integrate atomic trees into policy trees. In constructing a policy tree, we take into consideration the predictable student actions and the corresponding system actions. As described, we observed that student questions in a tutoring session likely concern concepts having prerequisite/successor relationships. Thus in a session, we can predict possible student questions.

For each concept in the subject, we create an atomic tree. In the atomic tree of C, the root is $(!C)$, and there is one or more edges for connecting the root with its children. The edges are labelled with the predictable student actions after the root action is taken: After $(!C)$, the predictable student action can be an *acceptance* action, or a question about one of the prerequisites of C. We thus label the edges with the predictable student actions (treated as observations).

Assume C has L $(L = 0, 1, 2, ...)$ direct successors, and M $(M = 0, 1, 2, ...)$ direct and indirect prerequisites. In the atomic tree of C, the root has $M+1$ edges connecting it with as many as $M+1$ children, which are expanded atomic trees of other concepts. The last edge is labeled (Θ) for connecting the root with the atomic tree of one of the direct successors of C. When $L = 0$, the edge connects the root with an action for terminating the session.

When $M > 0$, the first M edges are for connecting the root with the atomic trees of the M prerequisites. The ith edge is labeled with $(?C_i)$ and connects the atomic tree of C_i, which is the ith prerequisite of C $(1 \leq i \leq M)$. When $M = 0$, the atomic tree has only one edge connecting a successor. Figure 2 illustrates the atomic trees of ML (machine language, top), PL (programming language, bottom left), BD (binary digit, bottom middle), and IN (instruction, bottom right), based on the DAG in Figure 1.

In an atomic tree, a triangle with a concept name is a *dummy subtree*, and "Suc" stands for a successor. A dummy subtree with name C is not a part of the atomic tree but will be substituted with the atomic tree of C or substituted with a system action for terminating the session, when the atomic tree is integrated into a policy tree.

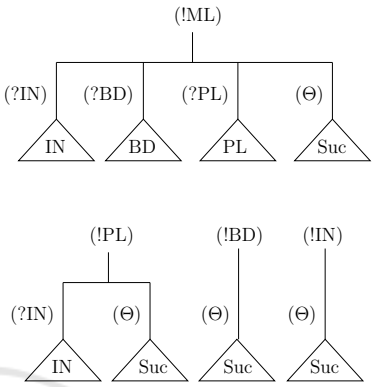

Figure 2: Atomic trees of ML, PL, BD, and IN.

6.3 Policy Tree Construction

Now we informally discuss the algorithm for constructing policy trees. We still denote the original and current questions by $(?C^o)$ and $(?C^u)$ respectively. We also assume C^u is in $(\wp_{C^o} \cup C^o)$, otherwise, we consider $(?C^u)$ starts a new session. For the pair of $(?C^o)$ and $(?C^u)$, we construct a set of policy trees $\mathcal{T}_{C^u}^{C^o}$. In the set there is one or more policy trees for each $C \in (\wp_{C^u} \cup C^u)$.

In a policy tree for C with C^o being the concept in the original question, the root action is $(!C)$, and every leaf is an action for terminating the session. The terminating action is connected by an edge of (Θ) to an action of $(!C^o)$. The connected $(!C^o)$ and (Θ) represent that the student accepts the answer to the original question. Thus the tutoring session can be terminated.

In a policy tree, every path from the root to a leaf is a process of tutoring. It starts with answering the current question, and ends when the student accepts the answer to the original question. Figure 3 illustrates a policy tree, in which the original and current questions are both (?ML). In this tree, a thick horizontal line denotes a terminating action.

We start constructing a policy tree for C in $\mathcal{T}_{C^u}^{C^o}$ by expanding the atomic tree of C: Substituting the dummy subtrees with atomic trees, then expanding the atomic trees, and so on, until all the leaf nodes become terminating actions. For example, in expand-

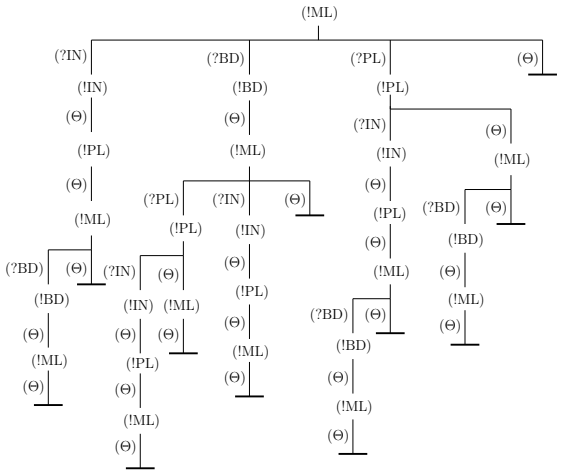

Figure 3: A policy tree for ML.

ing the atomic tree of ML, we substitute the dummy subtree named IN with the atomic tree of IN, then expand the atomic tree of IN by substituting the dummy subtree named Suc with the atomic tree of PL, which is a successor of IN, and so on, when we use a depth-first method.

In expanding an atomic tree, we have two rules for adding a terminating action and eliminating an edge:

- If the root of the atomic tree is $(!C^o)$, i.e. to answer the original question, substitute the dummy subtree connected by the edge of (Θ) with a terminating action.
- If the question associated with an edge has been answered in the path from the root, eliminate the edge without substituting the dummy subtree it connects.

The question $(?C')$ associated with an edge has been answered, if in the path from the root to the edge, there is a node of $(!C')$ or $(!C'')$ immediately followed by an edge of (Θ), where C'' is a direct or indirect successor of C'.

As an example, the set of $\mathcal{T}_{ML}^{ML}$ includes the policy trees for ML, IN, BD, and PL. (Note $(\wp_{ML} \cup ML) = \{\text{ML, IN, BD, PL}\}$.) The policy tree for ML is showed in Figure 3, the policy tree for IN is actually the first (left-most) subtree, the policy tree for BD is the second subtree, and the policy tree for PL is the third subtree.

6.4 Making a Decision with the Trees

All the sets of policy trees are pre-constructed and stored in a tree database. When the agent needs a strategy to answer a question, it retrieves the database, and gets a set of trees to evaluate to find the optimal. For example, when the original and current questions are both (?ML), the agent evaluates the four trees in $\mathcal{T}_{ML}^{ML}$ discussed above. In general, when the original question is $(?C^o)$ and the agent needs to answer the current question $(?C^u)$, it evaluates all the policy trees in $\mathcal{T}_{C^u}^{C^o}$ based on its belief, and finds the tree of the highest value (optimal tree). When choosing the optimal policy tree by using Eqn (6), we substitute $\mathcal{T}$ with $\mathcal{T}_{C^u}^{C^o}$.

A policy tree is not a tutoring plan that the agent must follow in the future. It is the strategy for the current step. After the optimal tree is selected, the agent takes the root action. After taking the action, it terminates the session or has a new current question, depending on the student action (observation):

- If the student action is (Θ), and the (Θ) edge connects to a terminating action, the agent terminates the tutoring session;
- If the student action is (Θ), and the (Θ) connect to $(!C)$, the agent considers $(?C)$ as the current question in the next step.
- If the student action is $(?C)$, the agent considers $(?C)$ as the current question in the next step.

In the next step, to answer the current question which is determined by using the rule given above, the agent chooses an action in the same way, i.e. by evaluating a set of policy trees, and so on. Continue the above example with (?ML) being both the original and current questions. If the policy tree for ML is the optimal, the agent takes action (!ML). After (!ML), if the student action is (Θ), the agent follows the edge of (Θ) in the tree, and takes the terminating action to finish the session. However, if after (!ML) the student action is (?PL), the agent considers the current question in the next step is (?PL). It evaluates the trees in $\mathcal{T}_{PL}^{ML}$, and continues until it takes a terminating action.

7 EXPERIMENTS AND ANALYSIS

We conducted the experiments by using the experimental ITS described before, which teaches concepts in software basics. Currently, keyboard and screen are used for system input and output. It interactively teaches a student at a time, answering the student's questions about the concepts. When the student asks a question about a concept that has prerequisites, the system chooses an optimal strategy to teach. POMDP is the engine to make decisions based on information (belief) about the student's current knowledge states.

We use "rejection rate" to evaluate how the students like the system's answers. After the system explains a concept, if the student asks a question about a prerequisite of the concept, or says something like

"I already know this concept", we consider that the student rejects the system action. The rejection rate is defined as the ratio of the number of rejected system actions to the total number of system actions.

Encouraging results have been achieved. Compared with directly teaching the concept asked by the student, or randomly choosing a prerequisite to start with, the teaching based on students' knowledge states has achieved better result. The rejection rates have dropped significantly. We do not discuss the performance in adaptive teaching, for this paper mainly addresses the problem of solution space.

Table 1: Numbers of concepts, tree sets, trees, and tree heights in sub-spaces.

Sub-space	# of concepts	# of tree sets	# of trees	Max height
1	21	231	1,771	30
2	23	276	2,300	18
3	20	210	1,540	26
4	27	378	3,654	35
5	25	325	2,925	32
6	26	378	3,744	37

The data set used to generate the following results includes about 90 concepts in software basics. The concepts have zero to five direct prerequisites. Based on the prerequisite/successor relationships, we partition the state space into six sub-spaces (Wang, 2015). Also based on the prerequisite/successor relationships, the policy tree construction algorithm created six groups of policy trees. Table I lists the numbers of concepts, numbers of tree sets, numbers of policy trees, and maximum tree heights in the subspaces. The construction of policy trees does not rely on the result of state space partitioning. Since both space partitioning and tree construction are based on concept prerequisite/successor relationships, the concepts in a policy tree are in the same subspace. Thus we can consider that the solution space is split into the same number of subspaces.

In the experiments, the average number of policy trees (τs) in a tree set ($\mathcal{T}$) is less than ten. The average height of the policy trees is less than 20. The maximum number of edges at a node is the number of concepts in a subspace plus one (the acceptance action). As we observed, the actual numbers of edges at nodes are much smaller than the numbers of concepts in subspaces. Many edges are eliminated in policy tree construction (as described in the subsection of tree construction.). In a decision step, the number of policy trees to evaluate, the heights of the trees, and the numbers of edges at nodes depend on the concept in the current question. For a concept near the lower end (having less prerequisites) the three numbers are small. For a concept near the higher end (having more prerequisites) the three numbers are big. For the higher end concepts, there is room for further improvement.

When making a decision, the agent evaluates a small number of trees. The average is less than 10 in the experiments. This does not create major efficiency problems for a modern computer. When the experimental ITS runs on a desktop computer with an Intel Core i5 3.2 GHz 64 bit processor and 16GB RAM, the response time for answering a question is less than 300 milliseconds. This includes the time for calculating a new belief, choosing a policy tree, and accessing the database of domain model. For a tutoring system, such response time could be considered acceptable.

The experiments show that the policy tree technique can minimize the number of policy trees evaluated in making a decision. When looking for a strategy to answer a question, the agent needs to evaluate a set of pre-constructed policy trees, determined by the original and current questions. The set includes only the policy trees for the concepts in the current question and some of its prerequisites and successors. They are the concepts that the student may want to learn after asking the original question. The policy trees for the concepts unrelated to the tutoring session are not included in the set. In addition, since the policy trees to evaluate include the related concepts only, and unnecessary edges are pruned as early as possible, the costs for evaluating individual trees have been reduced.

8 CONCLUDING REMARKS

Policy trees have been accepted as a practical approach for POMDP solving. In developing a policy tree approach, two key tasks are minimizing the number of trees to evaluate in making a decision, and the costs for evaluating individual trees. So far, the tasks have not been well researched in building POMDP based ITSs. In our research, we contribute a new technique for constructing policy trees, and applying them in choosing optimal tutoring actions. Our technique is based on the nature of education processes, and thus especially suitable for building systems of computer supported education. The technique can be applied to any tutoring tasks in which the subjects have pedagogical orders in the contents. Encouraging results have been achieved in our initial experiments.

ACKNOWLEDGEMENTS

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

REFERENCES

Bloom, B. S. (1984) The 2 sigma problem: the search for methods of group instructions as effective as one-to-one tutoring. In *Educational Researcher*, 13(6), 4-16.

Carlin, A. and Zilberstein, S. (2008) Observation Compression in DEC-POMDP policy trees. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multi-agent Systems*, 31-45.

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.

Cheung, B., Hui, L., Zhang, J., and Yiu, S. M. (2003) SmartTutor: an intelligent tutoring system in web-based adult education. In *Elsevier The journal of Systems and software*, 68, 11-25.

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

Folsom-Kovarik, J. T., Sukthankar, G., and Schatz, S. (2013) Tractable POMDP representations for intelligent tutoring systems. In *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.

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

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

VanLehn, K., van de Sande, B., Shelby, R., and Gershman, S. (2010) The Andes physics tutoring system: an experiment in Freedom. In Nkambou et-al eds. *Advances in Intelligent Tutoring Systems*. Berlin Heidelberg: Springer-Verlag, 421-443.

Wang, F. (2015) Handling Exponential State Space in a POMDP-Based Intelligent Tutoring System. In *Proceedings of 6th International Conference on E-Service and Knowledge Management (IIAI ESKM 2015)*, 67-72.

Williams, J. D., Poupart, P., and Young, S. (2005) Factored Partially Observable Markov Decision Processes for Dialogue Management. In *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. In *Elsevier Computer Speech and Language*, 21, 393-422.

Woolf, B. P. (2009) *Building Intelligent Interactive Tutors*. Burlington, MA, USA: Morgan Kaufmann Publishers.