Dynamical Creation of Policy Trees for a POMDP-based Intelligent Tutoring System

Fangju Wang
University of Guelph, 50 Stone Road East, Guelph, Ontario, Canada

Keywords: Intelligent Tutoring System, Computer Supported Education, Partially Observable Markov Decision Process, Computational Complexity.

Abstract: In this paper, we discuss a new technique for creating policy trees in an intelligent tutoring system (ITS) that is based on a partially observable Markov decision process (POMDP). The POMDP model is a useful tool for dealing with uncertainties. With a POMDP, an ITS may choose optimal teaching actions even when uncertainties exist. Great computational complexity in solving a POMDP has been a major obstacle to applying the POMDP model to intelligent tutoring. The technique of policy trees is considered a less expensive approach. However, policy trees are still too expensive for building ITSs that teach practical subjects. In our research, we develop a new technique of policy trees, in which trees are grouped and dynamically created. This technique has advantages of better time and space efficiencies. It enables us to build more efficient ITSs. Particularly the technique makes it possible to build ITSs on platforms which have limited storage capacity and computing power.

1 INTRODUCTION

Computational complexity is a key issue in building an interactive intelligent tutoring system (ITS). An ITS must be able to reside on a computing platform, and respond to student questions or requests in a timely fashion. However, many mathematical models underlying ITSs are computationally intractable. Huge space consumption and lengthy computing time have been major obstacles to applying the models to intelligent tutoring. The partially observable Markov decision process (POMDP) model is one of them.

The POMDP model may enable an ITS to choose optimal actions in teaching a student, even when uncertainties exist. A major goal for building an ITS is adaptive teaching. To be adaptive, in each tutoring step a system should be able to choose the action that is most beneficial to the student it teaches. Mathematically, adaptive tutoring can be modeled by a *Markov decision process (MDP)*, in which the agent makes optimal decisions considering the current states. In an MDP, the agent observes states clearly and knows exactly what the current states are. However, in a tutoring process, the teacher is often uncertain about student states (Woolf, 2009).

A POMDP is an extension of an MDP for dealing with uncertainties. In a POMDP, the task of choosing an optimal action is referred to as *solving the POMDP*. This task is computationally expensive. A simplified, less expensive technique for POMDP-solving is to use *policy trees*, in which decision making involves evaluating a set of trees and choosing an optimal one. However, the technique of policy trees is still too expensive to be used in practical applications. In making a decision, the number of trees to evaluate is exponential, and the number of operations in evaluating a tree is also exponential. To apply the POMDP model to intelligent tutoring, we must address the problems of computational complexity.

In our research, we develop a new technique of policy trees for POMDP-solving in an ITS. In this technique, policy trees are subdivided into small tree sets. When making a decision, the POMDP agent evaluates the trees in a set, instead of all the possible trees. A tree set is dynamically created when the agent needs to evaluate it. The technique has advantages of better time and space efficiencies.

In this paper, we first introduce the technical background of the POMDP model that is needed for discussing our technique, followed by reviewing the existing work related to our research. Then we present our techniques for grouping trees and dynamic tree creation, and discuss some experimental results.

Wang, F.
Dynamical Creation of Policy Trees for a POMDP-based Intelligent Tutoring System.
DOI: 10.5220/0006774601370144
In *Proceedings of the 10th International Conference on Computer Supported Education (CSEDU 2018)*, pages 137-144
ISBN: 978-989-758-291-2

2 PARTIALLY OBSERVABLE MARKOV DECISION PROCESS

2.1 MDP and POMDP

A POMDP is an extension of an MDP, as mentioned in the previous section. An MDP can model a decision process in which different actions can be chosen in different states to maximize rewards. The core of an MDP includes S, A, and ρ, which are a set of states, a set of actions, and a reward function. In a decision step, the agent is in $s \in S$, takes $a \in A$ that is available in s, enters $s' \in S$, and receives reward $\rho(s,a,s')$. The MDP model is stochastic. An MDP includes T, which is a set of state transition probabilities. $P(s'|s,a) \in T$ is the probability that the agent enters s' after taking a in s. Another core component in an MDP is *policy* $\pi(s)$. It guides the agent to choose the optimal action available in s to maximize rewards.

As an extension of an MDP, a POMDP has two additional core components: O and Z, which are a set of observations and a set of observation probabilities. A POMDP can model a decision process in which the agent is not able to see states completely. In a POMDP, the agent infers information about states and represents the information about states by a *belief*, denoted by b. In a decision step, the agent is in $s \in S$ that it is not able to see, chooses $a \in A$ based on its current belief b, enters $s' \in S$ that it is not able to see either, observes $o \in O$, and infers information about s' by using $P(o|a,s') \in Z$ and $P(s'|s,a) \in T$.

Belief b is defined as

$$b = [b(s_1), b(s_2), ..., b(s_Q)] \tag{1}$$

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 $\sum_{i=1}^{Q} b(s_i) = 1$.

In a POMDP, the policy is $\pi(b)$. In a decision step, it guides the agent to choose an action considering the current belief b to maximize the long term reward.

2.2 Policy Trees for POMDP-Solving

For a given b, an optimal π returns an optimal action. In a POMDP, finding the optimal π is referred to as *solving the POMDP*. For most practical application problems, POMDP-solving is a task of great complexity (Carlin and Zilberstein, 2008; Rafferty et al., 2011). A simplified, less expensive technique for POMDP-solving is to use *policy trees*.

In a policy tree, nodes are actions, and edges are observations. The action at the root is called the *root action*. An action node has observation edges to actions at the next level. After an action is taken, the next action to take is one of the actions at the next level, depending on what the agent observes. Figure 1 illustrates the general structure of a policy tree, in which a_r is the root action, a is an action, and K is the number of possible observations. Note that an action node has edges of *all* the possible observations to the next level.

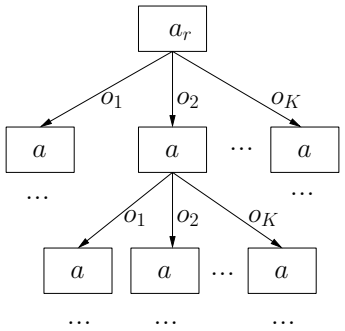

Figure 1: The general structure of a policy tree.

When a technique of policy trees is used for POMDP-solving, finding the optimal policy is to identify the optimal tree. In each decision step, the agent finds the optimal policy tree considering its current belief, and takes the root action of the tree. In the following, we discuss a method to find the optimal tree.

Each policy tree is associated with a *value function*, which evaluates the long term reward of taking the tree (policy). Let τ be a policy tree. The value function of state s given τ is

$$V^{\tau}(s) = \mathcal{R}(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \sum_{o \in O} P(o|a,s') V^{\tau(o)}(s') \tag{2}$$

where a is the root action of τ, s' is the next state, i.e. the state that the agent enters into after taking a, γ is a discounting factor ($0 \leq \gamma \leq 1$), o is the observation after a is taken, $\tau(o)$ is the subtree in τ which is connected to the root by the edge of o, and $\mathcal{R}(s,a)$ is the expected immediate reward after a is taken in s, calculated as

$$\mathcal{R}(s,a) = \sum_{s' \in S} P(s'|s,a) \mathcal{R}(s,a,s') \tag{3}$$

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 (2) is the discounted expected value.

From Eqns (1) and (2), we have the value function of belief b given τ:

$$V^{\tau}(b) = \sum_{s \in S} b(s) V^{\tau}(s). \tag{4}$$

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

$$\pi(b) = \hat{\tau} = \arg\max_{\tau \in \mathcal{T}} V^{\tau}(b), \quad (5)$$

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

The size of a policy tree depends on the number of possible observations and the horizon. When the horizon is H, the number of nodes in a tree is

$$\sum_{t=0}^{H-1} |O|^t = \frac{|O|^H - 1}{|O| - 1} \quad (6)$$

where $|O|$ is the size of O. At each node, the number of possible actions is $|A|$. Therefore, the total number of all possible H-horizon policy trees is

$$|A|^{\frac{|O|^H - 1}{|O| - 1}}. \quad (7)$$

Both numbers are exponential.

3 RELATED WORK

Researchers in the fields of ITSs have seen the great potential of the POMDP model in building ITSs. Extensive research has been conducted in applying POMDPs to intelligent tutoring (Cassandra, 1998; Williams et al., 2005; Williams and Young, 2007; Theocharous et al., 2009; Rafferty et al., 2011; Chinaei et al., 2012; Folsom-Kovarik et al., 2013). In the work related to applying the model to ITSs, POMDPs were used to model student states, and to customize and optimize teaching. In a commonly used structure, student states had a boolean attribute for each of the subject contents, actions available to a tutoring agent were various types of teaching techniques, and observations were results of tests given periodically. The goals were to teach as many of the contents in a finite amount of time, or to minimize the time required to learn the entire subject. In the following, we review some work in which policy trees were used for POMDP-solving in ITSs.

Rafferty and co-workers created a POMDP-based system for teaching concepts (Rafferty et al., 2011). A core component of the system was a technique of fast 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, and the transitions modeled how teaching actions stochastically changed the learners' knowledge.

For solving the POMDP, the researchers developed a 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, and limited the horizon to control the depth of the tree.

In the work reported in (Wang, 2016), an experimental ITS was developed for teaching concepts in computer science. A POMDP was used in the system to model processes of intelligent tutoring. In the POMDP, states, actions, and observations modeled student knowledge states, system tutoring actions, and student actions, respectively. A method of policy trees was proposed for POMDP-solving. In the method, policy trees were created and stored in a tree database. To choose an optimal action to respond to a given student query, the agent searched the database and evaluated a set of trees. For reducing the costs in making a decision, techniques were developed to minimize the tree sizes and decrease the number of trees to evaluate. A major disadvantage of the proposed policy tree method was its space complexity.

The techniques of policy trees for improving POMDP-solving have made good progress towards building practical POMDP-based ITSs. However, they were still too costly to use. For example, as the authors of (Rafferty et al., 2011) concluded, computational challenges existed in their technique of forward trees, despite sampling only a fraction of possible actions and using short horizons. Also, how to sample actions and how to shorten a horizon are challenging problems. Computational complexity has been a bottleneck in applying the POMDP model to intelligent tutoring.

4 GROUPING OF POLICY TREES

4.1 An Overview

To address the problems of computational complexity in applying the method of policy trees in a POMDP-based ITS, we develop a new technique, in which policy trees are grouped and trees are dynamically cre-

ated. With this technique, the agent evaluates a small set of trees when making a decision. The trees in the set are dynamically created for better space efficiency. In this section, we discuss the grouping of trees, and in the next section, the dynamic tree creation.

The discussion is in the context of an experimental ITS, which we developed for testing our techniques. The ITS teaches concepts in software basics. A POMDP helps the ITS choose optimal teaching actions. We cast the ITS on to the POMDP, by using POMDP states to represent student knowledge states, and actions to represent system tutoring actions. We treat student actions (asking questions, accepting answers, etc) as observations.

4.2 State Space Partitioning

To have small tree sets, we partition the state space into subspace, and then group trees in each subspace into tree sets. Before the tree grouping technique, we discuss our method for state space partitioning.

We define the states in terms of concepts in the instructional subject. In software basics, concepts include *program*, *instruction*, *algorithm*, and many others. We associate each state with a *state formula*, which is of the form:

$$(C_1 C_2 C_3 ... C_N), \tag{8}$$

where C_i is the variable for the ith concept C_i, taking a value $\surd C_i$ or $\neg C_i$ $(1 \leq i \leq N)$, and N is the number of concepts in the subject. We use $\surd C_i$ to represent that the student understands C_i, and $\neg C_i$ to represent that the student does not. A formula is a representation of a student knowledge state. For example, formula $(\surd C_1 \surd C_2 \neg C_3 ...)$ is a representation of the state in which the student understands C_1 and C_2, but not C_3, ... States thus defined have Markov property.

When there are N concepts in an instructional subject, the number of state formulae is 2^N. This implies that the number of possible states is 2^N. As can be seen in Eqn (2), the cost for evaluating a value function is proportional to the size of state space. To reduce the cost, we partition the state space into smaller subspaces. The partitioning technique is based on prerequisite relationships between concepts.

Prerequisite relationships are pedagogical orders of concepts. A concept may have zero or more prerequisites, and a concept may serve as a prerequisite of zero or more concepts. For example, in mathematics, *derivative* has prerequisites *function*, *limit* and so on, and *function* is a prerequisite of *derivative*, *integral*, and so on. To understand a concept well, a student should understand all its prerequisites first. For a set of concepts, the prerequisite relationships can be represented by a directed acyclic graph (DAG). In this paper, when a concept is a prerequisite of another, we call the latter a *successor* of the former.

In the first step of our space partitioning technique, we subdivide concepts such that concepts having prerequisite relationships are in the same group. Some very "basic" concepts may be in two or more groups. In the second step, for each group, we create a state subspace by using concepts in the group to define states, in the way just discussed. In the third step, we eliminate invalid states. For details of the partitioning technique, please see (Wang, 2015). After space partitioning, we create policy trees for each subspace. In a tree, the nodes and edges concern concepts in the subspace only.

This partitioning technique is based on our observation that in a window in a tutoring process, student questions likely concern concepts that have prerequisite relationships with each other. The observation suggests that we could localize the computing for choosing an optimal teaching action within a smaller state subspace defined by concepts having prerequisite relationships.

The total number of states in the subspaces are much smaller than the number of states in the space defined by using all the concepts. In addition, the number and sizes of policy trees in subspaces are much smaller because the sets of actions and observations are smaller.

4.3 Policy Tree Grouping

The cost for making a decision depends on the number of trees to evaluate, that is, the size of $\mathcal{T}$ in Eqn (5). For lower costs, we group the trees in each subspace into small tree sets. When choosing an optimal teaching action, the agent evaluates trees in a single set. For discussing the grouping method, we first define *optimal action* and *tutoring session*.

In science and mathematics subjects, many concepts have prerequisites. When the student asks about a concept, the system should decide whether it would start with teaching a prerequisite for the student to make up some required knowledge, and, if so, which one to teach. The *optimal* action is to teach 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 other concepts.

A *tutoring session* is a sequence of interleaved student and system actions, starting with a question about a concept, possibly followed by answers and questions concerning the concept and its prerequisites, and ending with a student action accepting the

answer to the original question. If, before the acceptance action, the student asks a concept that has no prerequisite relationship with the concept originally asked, we consider that a new tutoring session starts.

We classify questions in a session into the *original question* and *current questions*. The original question starts the session, concerning the concept the student originally wants to learn. We denote the original question by $(?C^o)$, where C^o is the concept concerned in the question and superscipt o stands for "original". A current question is the question to be answered by the agent at a point in the session, usually for the student to make up some knowledge. A current question may be asked by the student, or made by the agent. We denote a current question by $(?C^c)$, where C^c is the concept concerned in the question, and superscipt c stands for "current". Concept C^c is in $(\wp_{C^o} \cup C^o)$, where $\wp_{C^o}$ is the set of all the direct and indirect prerequisites of C^o.

In the following, we discuss an example, which involves concepts *database* and *file*. We assume that *file* is a prerequisite of *database*. At a point in a tutoring process, the student asks question "What is a database?" If *database* has no prerequisite relationship with the concepts asked/taught right before the question, we consider the question starts a new tutoring session, and it is the original question of the session. If the agent believes that the student already understands all the prerequisites of *database*, and answers the question directly, the question is also the current question when the agent answers it. If the agent teaches *database* in terms of *file*, and then the student asks question "What is a file?", the system action of teaching *database* is not an optimal because the student needs to make up a prerequisite. At this point the question about *file* is the current question. If the agent answers the question about *file* and the student satisfies the answer, the system action is optimal.

Now we consider the grouping of trees. When the agent has current question $(?C^c)$ to answer, it needs to choose an optimal action. The optimal action may be to teach C^c or teach one of the prerequisites of C^c, depending on the agent's belief about the student's knowledge state. Recall that in a tutoring step, the agent evaluates a set of trees and chooses the root action of the tree that has the highest value. The set of trees to evaluate to answer $(?C^c)$ should include trees in which root actions are to teach C^c or prerequisites of C^c. Since the ultimate goal to answer $(?C^c)$ is to answer the original question $(?C^o)$, actions to answer $(?C^o)$ should be included in the trees in the set.

Based on the above consideration, we have our grouping strategy: for each possible pair of $(?C^o)$ and $(?C^c)$, we create tree set $\mathcal{T}_{C^c}^{C^o}$. In a tutoring session with original question $(?C^o)$, to choose an optimal action to answer current question $(?C^c)$, the agent evaluates trees in $\mathcal{T}_{C^c}^{C^o}$. Since $\mathcal{T}_{C^c}^{C^o}$ is normally much smaller than the set of all the possible trees, the cost for choosing an optimal tree can be significantly reduced. The tree structure will be discussed in the next section.

5 DYNAMIC CREATION OF POLICY TREES

5.1 Structure of the Policy Trees

In the following, we denote the system action for teaching concept C by $(!C)$, and denote a student acceptance action by (Θ). An acceptance action can be something like "I understand", and "I see".

As just discussed, in a state subspace, for each possible pair of original and current questions, denoted by $(?C^o)$ and $(?C^c)$, we create tree set $\mathcal{T}_{C^c}^{C^o}$. The optimal action to answer current question $(?C^c)$ may be to teach C^c, or to teach one of the prerequisites of C^c. Therefore, the trees in $\mathcal{T}_{C^c}^{C^o}$ have root actions $(!C^c)$, or $(!C)$ where $C \in \wp_{C^c}$. For each $C \in \wp_{C^c}$, we have a tree in $\mathcal{T}_{C^c}^{C^o}$, of which the root is $(!C)$. We denote a tree in $\mathcal{T}_{C^c}^{C^o}$ with root action $(!C)$ by $\mathcal{T}_{C^c}^{C^o}.\tau_C$.

In a tutoring session started by $(?C^o)$, the ultimate goal of the agent is to teach C^o. In a policy tree for answering a current question, any leaf node must be a system action to terminate the session, after a student acceptance action that accepts $(!C^o)$. A path in the tree includes possible questions and answers concerning prerequisites of C^o. Also, since possible student questions in the session concern prerequisites of C^o, we limit the observation set O to questions concerning concepts in $\wp_{C^o}$ only. (Student actions are treated as observations.)

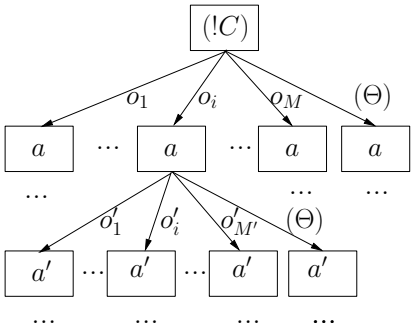

Figure 2: The general structure of policy tree $\mathcal{T}_{C^c}^{C^o}.\tau_C$.

Figure 2 illustrates the general structure of policy tree $\mathcal{T}_{C^c}^{C^o}.\tau_C$. The root of $\mathcal{T}_{C^c}^{C^o}.\tau_C$ is $(!C)$, i.e. an

action for teaching $C \in (\wp_{C^c} \cup C^c)$. When C has M prerequisites C_1, ... C_M, the root has $M+1$ children. The observations $o_1, ..., o_i, ..., o_M$ are student actions $(?C_1), ..., (?C_i), ..., (?C_M)$. The first M children are sub-trees connected by the observation edges. Actions at the sub-tree roots are $(!C_1), ..., (!C_M)$. Note that edge o_i $(1 \leq i \leq M)$ may connect to any one of them. The last child is a sub-tree rooted by $(!C^u)$, where C^u is one of the direct *successors* of C. When C has more than one direct successor, C^u is the one on the path from C^o to C. The semantics of such root-children structure is that after $(!C)$, if the student accepts $(!C)$, teach the direct successor C^u, if the student asks about a prerequisite of C, teach a prerequisite. The prerequisite to teach is dynamically selected. The selection will be discussed in the next subsection.

In a policy tree, each sub-tree is structured in the same way. That is, the root has edges for prerequisites of the concept in the root action and an acceptance edge, illustrated by o'_1, ..., o'_i, ..., $o'_{M'}$ and (Θ) in Figure 2, where M' is the number of prerequisites of the concept in the action connected by o_i. However, if a prerequisite has been taught in the path from the tree root, the edge is not included. If a root is $(!C^o)$ for answering the original question, its acceptance edge connects to an action terminating the session.

5.2 Creation of the Policy Trees

When the student asks question $(?C^o)$ starting a new tutoring session, the agent goes to the subspace that contains C^o, and contains all the prerequisites of C^o as well. To answer current question $(?C^c)$ in the session, it evaluates all the trees in tree set $\mathcal{T}_{C^c}^{C^o}$. We have developed a new technique to dynamically create the tree set when it is evaluated. This technique has better space efficiency than the method of storing a tree database.

As discussed in the previous subsection, in trees in $\mathcal{T}_{C^c}^{C^o}$, root actions teach concepts in $(\wp_{C^c} \cup C^c)$. In the general structure of a policy tree (illustrated in Figure 1), each edge (observation) connects to all the possible actions (in different trees). With this structure, for each $C \in (\wp_{C^c} \cup C^c)$ the number of trees with root $(!C)$ is exponential in the number of possible observations (see Eqn (7)). That is, the number of trees having the same root action is exponential.

To reduce the cost for evaluating policy trees, in $\mathcal{T}_{C^c}^{C^o}$ we create only one tree for each $C \in (\wp_{C^c} \cup C^c)$, and use it to approximate an exponential number of trees. To have only one tree for each $C \in (\wp_{C^c} \cup C^c)$, we connect each edge to one action, instead of all the possible actions. For example, when creating the tree in Figure 2, we select one action for edge o_1, one action for o_2, ... one action for o'_1, and so on.

In our research, we discovered that in a state only a small number of actions have large enough chances to be taken. In computing Eqn (2) for evaluating trees, most actions contribute little to tree values. This suggests that we would not lose much information when ignoring the actions that are less likely to be taken and contribute less. In the following, we discuss the selection of an action for each edge by using the tree in Figure 3.

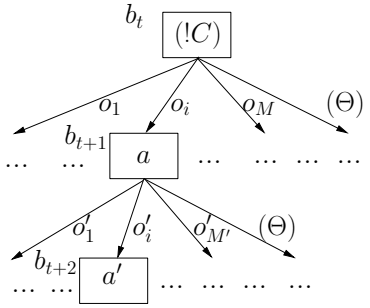

Figure 3: The general structure of policy tree $\mathcal{T}_{C^c}^{C^o}.\tau_C$.

Assume at time step t, the agent has belief b_t, and will evaluate tree set $\mathcal{T}_{C^c}^{C^o}$, and assume the tree in Figure 3 is $\mathcal{T}_{C^c}^{C^o}.\tau_C$. In creating the tree, we select $(!C)$ as the root, which is a possible action to take at t, and then select an action for each edge based on an updated belief at the next level, and so on. For example, we need to select action a for edge o_i based the updated belief b_{t+1}, select a' for o'_i based on the updated b_{t+2}, and so on.

A belief is a set of probabilities (see Eqn (1)). To update a belief, we update each of the probabilities. The following is the formula to calculate element $b'(s')$ in updated belief b':

$$b'(s') = \sum_{s \in S} b(s) P(s'|s,a) P(o|a,s') / P(o|a) \qquad (9)$$

where $P(s'|s,a) \in T$ and $P(o|a,s') \in Z$ transition probability and observation probability, $P(o|a)$ is the total probability for the agent to observe o after a is taken, calculated as

$$P(o|a) = \sum_{s \in S} b(s) \sum_{s' \in S} P(s'|s,a) P(o|a,s'). \qquad (10)$$

$P(o|a)$ is used in Eqn (9) as a normalization. Using Eqn (9) we can calculate b_{t+1} from b_t, $(!C)$, and o_i.

Let b_{t+1} be

$$b_{t+1} = [b_{t+1}(s_1), b_{t+1}(s_2), ..., b_{t+1}(s_Q)]. \qquad (11)$$

In b_{t+1} we can find the j such that $b_{t+1}(s_j) \geq b_{t+1}(s_k)$ for all the $k \neq j$ $(1 \leq j,k \leq Q)$. Assume the state formula of s_j is

$$(\sqrt{C_1}\sqrt{C_2}...\sqrt{C_{l-1}}\neg C_l ... \neg C_{N'}). \qquad (12)$$

The belief and state formula indicate that most likely the student does not understand C_l, but understands all of its prerequisites. Therefore, $\rho(s_j,(!C_l))$ returns a high value. Considering a single step, we select $(!C_l)$ as an optimal action at b_{t+1}. Thus we select $(!C_l)$ as a, and connect the edge of o_i to it.

6 EXPERIMENTAL RESULTS AND DISCUSSION

In this section, we present two sets of experimental results. The first set includes the results of evaluating adaptive teaching of the system, and the second set includes the results of testing the technique for dynamic creation of trees. The data set used in the experiments included 90 concepts in software basics. Each concept had zero to five prerequisites.

We used a two-sample *t*-test method to evaluate the system performance in adaptive teaching. The test method was the *independent-samples t-test*. 30 students participated in the experiment, who were adults without formal training in computing. The students were randomly divided into two equal size groups. Group 1 studied with the ITS with the POMDP turned off, and Group 2 studied with the POMDP turned on. Each student studied with the ITS for about 45 minutes. The student asked questions about concepts in the subject, and ITS taught the concepts. The performance parameter was *rejection rate*, which was the ratio of the number of system actions rejected by a student to the total number of system actions for teaching the student.

For each student, we calculated a rejection rate. For the two groups, we calculated mean rejection rates $\bar{X}_1$ and $\bar{X}_2$. The two sample means were used to represent population means μ_1 and 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 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 could reject H_0 and accept H_a. The analysis suggested that the POMDP could significantly reduce the rejection rate. This implies that the POMDP helped the system significantly improve adaptive teaching.

We tested the dynamic tree creation technique with the same data set of software basics, on a desktop computer with an Intel Core i5 3.2 GHz 64 bit processor and 16GB RAM. For comparison, we also tested a static tree creation technique. Both the static and dynamic tree creation techniques have the same algorithm for grouping trees. The difference is that in the static technique all the tree sets were created and stored in a database before the ITS started teaching students, while in the dynamic technique a tree set was created right before it was evaluated. In both techniques, the state space was subdivided into six subspaces. The largest subspace included 27 concepts, 4,970 valid states, 170 tree sets, and 688 trees.

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	$s_1^2 = 0.0158$	$s_2^2 = 0.0113$

Table 2: Comparison between static and dynamic tree creation methods.

	Static creation	Dynamic creation
Permanent pace usage	1.078GB	0
Max space usage	1.078GB	215.68MB
Database creation time	36,888ms	
Max tree creation time		158ms
Belief update time	525ms	518ms
Max decision time	147ms	152ms
Max Response time	669ms	828ms

In Table 2 we list results of the two techniques. The space usage includes that for the tree database or tree sets only. The maximum tree creation time was for creating the largest tree set, of which the size was 215.68MB. The maximum decision time was for evaluating trees in the largest tree set and choosing the optimal tree. Response time included the time for calculating a new belief, and evaluating a tree set to choose an optimal tree. The maximum response time was recorded when the largest tree set was evaluated.

The experimental results suggest that the dynamic tree creation technique is effective for building space-efficient ITSs. Its space usage was a small fraction of that of a static tree creation technique. In terms of time efficiency, the dynamic technique was comparable to a static technique. Since the time for dynamically creating a tree set is short, the total response time was only slightly longer than that of the static technique. As can be seen in Table 2, the maximum response time with dynamic tree creation was less than a second. For a tutoring system, such response time could be considered acceptable. Time efficiency can be improved with a cache of tree sets.

7 CONCLUDING REMARKS

We have developed a new technique to address the space complexity problem in building POMDP-based ITSs. In this technique, policy trees are dynamically created when they are evaluated, and no space is required to store a tree database. The technique is especially useful for building ITSs on handheld devices, which usually have limited storage spaces. While significantly improving space efficiency, the technique does not sacrifice much time efficiency. In some cases, it may even have advantages in time efficiency. For example, for system without durable storage, the technique may largely reduce the time to start, since the lengthy tree database creation is not needed.

ACKNOWLEDGEMENTS

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

REFERENCES

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*, pages 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*, pages 17–24.

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*, pages 280–286.

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.

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)*, pages 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*.

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)*, pages 67–72.

Wang, F. (2016). 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)*, pages 85–93.

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. *Elsevier Computer Speech and Language*, 21:393–422.

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