Developing Space Efficient Techniques for Building POMDP Based Intelligent Tutoring Systems

Fangju Wang

School of Computer Science
University of Guelph
Guelph, Ontario, Canada N1G 2W1
Email: fjwang@uoguelph.ca

***Abstract*—In building an intelligent tutoring system (ITS), the partially observable Markov decision process (POMDP) model provides useful tools to deal with uncertainties, which are major challenges in achieving adaptive teaching. However, the POMDP model is very expensive. When a method of policy trees is used in decision making, the number of trees and sizes of individual trees are typically exponential. The great space complexity obstructs application of the POMDP model to ITSs. In our research, we developed space efficient techniques to address the space complexity problem. The techniques minimize the number and sizes of trees, and reduce space consumption of the tree database. Encouraging results have been achieved: the techniques enabled us to build a system with a manageable size, to teach a practical subject.**

Keywords–Intelligent system; intelligent tutoring system; adaptive teaching; partially observable Markov decision process; space efficiency.

I. INTRODUCTION

In recent years, intelligent tutoring systems (ITSs) have been playing increasingly important roles in computer supported education, which is a remarkable development in education and training. ITSs have been built as teaching/learning aids, and has been beneficial to students and teachers in fields including mathematics [13], physics [8], computer science [13], Web based education [2], and military training [13].

A key feature of ITSs is adaptive teaching. In each tutoring step, an ITS should be able to take the optimal teaching action based on information about its student's current knowledge states. An ITS achieves adaptive teaching by tracing student knowledge states, and taking teaching actions based on the states. The core modules in an ITS include a domain model, a student model, and a tutoring model. The *domain model* stores the domain knowledge. The *student model* contains information about student states. The *tutoring model* represents the system's tutoring strategies.

Uncertainties in observing and tracing student states have been major difficulties in building adaptive teaching systems. Quite often, it is difficult to know exactly what the student's states are, and what the most beneficial tutoring actions should be [13]. The partially observable Markov decision process (POMDP) model provides useful tools for dealing with uncertainties. Recently, researchers have been applying POMDP techniques in building ITSs [6] [7].

A POMDP is an extension of a Markov decision process (MDP) for modeling processes in which decisions have to be made when uncertainties exist. In a POMDP, there is a state space, which is not completely observable. The decision agent infers its information about states based its actions and observations, and represents the information as a *belief*. In making a decision, it updates its belief, solves the POMDP for an optimal *policy*, and uses the policy to choose an action.

Great computational costs are primary obstacles to building a POMDP-based ITS. In a POMDP, both space and time complexities are typically exponential. To build a POMDP-based ITS for real world applications, we must address the problems of computational complexities. In earlier stages of our research, we developed techniques to reduce state spaces [9], and to minimize the numbers of policy trees that comprised solution spaces [10]. (The approach of policy trees is for POMDP solving. It will be discussed in details later.)

Although we have achieved progress, problems with space complexity are far from being solved. In an ITS for a practical subject, a policy tree database might become unmanageable in size and a single policy tree might exhaust the available memory space. In this paper, we report our new techniques for further reducing the size of a POMDP solution space in an ITS. The techniques were aimed at minimizing the sizes of individual trees.

In Section II, we review the work related to our research. In Sections III, IV and V, we briefly introduce the technical background of the POMDP model, and an ITS on POMDP, and discuss the space efficiency issues in a POMDP based ITSs. In Section VI, we describe our space efficient techniques, and in Section VII, present and analyze some experimental results.

II. RELATED WORK

The work of applying POMDP to computer supported education started in as early as 1990s [1]. In the early years, POMDP was used to model mental states of individuals, and to find the best ways to teach concepts. More recent work included [3] [4] [6] [7] [11] [12]. 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.

The technique of faster teaching by POMDP planning is the work reported in [6]. The technique was for computing approximate POMDP policies, with the goal to select actions to minimize the expected time for the learner to understand concepts. The researchers framed the process of choosing optimal

 ISBN: 978-1-61208-576-0

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.

The researchers developed a method of forward trees for solving the POMDP. Forward trees are variations of policy trees. For the current belief, a forward trees was constructed to estimate the value of each teaching 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 tree was constructed for selecting a new action. The costs for storing and evaluating a forward tree is exponential in the task horizon and the number of possible actions. To reduce the costs, the researchers restricted the trees by sampling only a few actions, and by limiting the horizon to control the sizes of trees.

In [4], a technique of gap elimination was developed to make POMDP solvers feasible for real-world problems. The researchers created a data structure to describe the current mental status of each 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 subject. Observations are indicators that particular gaps are present or absent. The intelligent tutor takes actions to discover and remove all gaps.

To deal with time and space efficiency problems, the researchers developed two scalable representations of states and observations: state queue and observation chain. 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 time and space efficiency in POMDPs have made good progress towards building ITSs for practical teaching. However they had limitations. For example, as the authors of [6] concluded, computational challenges still existed in the technique of forward trees, despite sampling only a fraction of possible actions and allowing very short horizons. Also, how to sample the possible actions and how to shorten the horizon are challenging problems. As the authors of [4] indicated, the methods of state queue and observation chain might cause information loss, which might in turn degrade system performance in choosing optimal actions.

III. Partially Observable Markov Decision Process

A POMDP consists of 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 reward $r = \rho(s, a, s')$. The probability of transition from 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 actions, 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 belief b, the optimal π returns the 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), the next action is determined by what is observed (at an edge). A path in a policy tree is a sequence of "action, observation, ..., action". Figure 1 illustrates a policy tree, where a_r is the root action, o_1, ..., o_K are possible observations, and a is an action. In a finite horizon POMDP of length H, a policy can be a tree of height H.

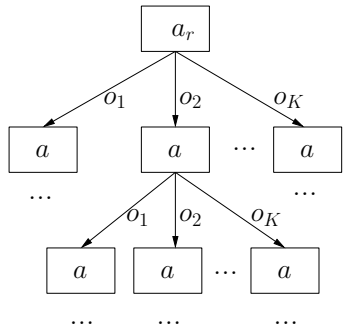

Figure 1. The general structure of a policy tree.

In the method of policy trees, making a decision is to choose the optimal tree and take its 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

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

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 (3) is the discounted expected value of future states.

 ISBN: 978-1-61208-576-0

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

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

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

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

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

From the above description, we can see that making a decision (by using (3), (4), (5), and (6)) requires computation over the entire state space S and solution space $\mathcal{T}$. The two spaces are typically exponential. They have been a bottleneck in applying POMDP to practical problems.

IV. An Intelligent Tutoring System on POMDP

We developed an experimental system as a test bed for our techniques, including the policy tree technique for intelligent tutoring. In this section, we describe how we cast an ITS onto the POMDP, and how we define states, actions, and observations.

The instructional subject of the ITS is basic knowledge of software. The system is for teaching concepts in the subject. It teaches 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.

Most concepts in the subject have prerequisites. When the student asks about a concept, the system decides whether it should 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.

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. 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.

We define states in terms of the concepts in the instructional subject. In software basics, the concepts are *data*, *program*, *algorithm*, and so on. We use a boolean variable to represent each concept: variable $\mathcal{C}_i$ represents concept C_i. $\mathcal{C}_i$ 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:

$$(\mathcal{C}_1 \mathcal{C}_2 \mathcal{C}_3 ... \mathcal{C}_N) \quad (7)$$

where $\mathcal{C}_i$ may take $\surd C_i$ or $\neg C_i$ ($1 \leq i \leq N$). We call a formula of (7) a *state formula*. It is a representation of which concepts the student understands and which concepts the students does not.

In an ITS for teaching concepts, 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", or "I am done". The system actions are mainly teaching concepts, like "A query language is a high-level language for querying." We use $(!C)$ to denote a system action of teaching C, and use (Φ) to denote a system action that does not teach a concept, for example a greeting. As mentioned, ITS actions are represented by POMDP actions, while student actions are treated as POMDP observations.

V. Addressing the Space Problems

A. The Space Problems

When states are defined in terms of concepts in the instructional subject, the number of state formulas is 2^N, where N is the number of concepts in the subject. When the method of policy trees is used for POMDP-solving, in a finite horizon POMDP of length H, the number of nodes in a policy tree is

$$\sum_{t=0}^{H-1} |O|^t = \frac{|O|^H - 1}{|O| - 1} \quad (8)$$

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

$$|A|^{\frac{|O|^H - 1}{|O| - 1}}. \quad (9)$$

The complexities result in great difficulties in creating and storing states and policy trees in memory. In the following, we report our techniques for addressing the space problems.

B. Prerequisite Relationships

We develop our techniques based on information about pedagogical orders for learning/teaching contents in instructional subjects. Prerequisite relationships between concepts in a subject are pedagogical orders of the concepts. If, to understand concept C_j the student must first understand concept C_i, C_i is referred to as a prerequisite of C_j. A concept may have zero or more prerequisites, and a concept may be a prerequisite of zero 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. Prerequisite relationships can be represented in a directed acyclic graph (DAG). Figure 2 illustrates a DAG representing direct prerequisite relationships in a subset of concepts in software basics.

We observed, through examining tutoring processes of human teachers and students, that concepts asked by a student in successive questions usually had prerequisite/successor

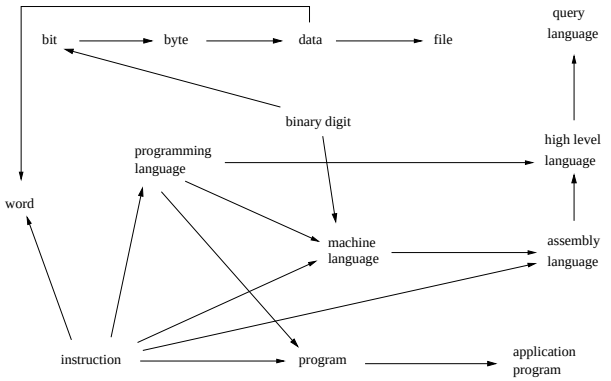

Figure 2. The DAG representing direct prerequisite relationships in a subset of the concepts in software basics. An arrow indicates "is a prerequisite of".

relationships with each other. Sometimes, after the teacher answered a question, the student asked about a prerequisite of the concept in the original question. This happened when the student realized that he/she needed to make up the prerequisite. Sometimes, after a teacher's answer, the student asked about a successor of the concept in the original question. This happened when the student had learned the concept and wanted to learn more along the line.

The observation suggests that we could group concepts in an instructional subject based on their prerequisite relationships, and limit computing within a subset of concepts when an ITS answers a question. We have developed a technique for partitioning a state space into sub-spaces and reducing the sizes of sub-spaces. The technique allows an ITS to localize the computing of a tutoring session within a sub-space. For details of the partitioning technique, please see [9]. In the next session, we describe the space efficient techniques for creating and storing policy trees.

VI. Space Efficient Techniques for Policy Trees

A. Design Consideration

In this section, we present our policy tree techniques for reducing the solution space of a POMDP, including design consideration, structures of solution space and policy trees, and decision making with the trees. We then describe a space saving structure, which enables creating large policy trees in limited memory. Our techniques can be applied to any ITSs, in which instructional subjects can be subdivided into small components to teach, and the components have pedagogical orders with each other.

It can be seen from (6), (8) and (9), when a technique of policy trees is applied, the costs (in time and space) for making a decision depend on the size of $\mathcal{T}$, the horizon H, and the sizes of S and O. The design goal of our techniques is to minimize the H, and the sizes of $\mathcal{T}$, S, and O that are involved in making a decision, with least loss of information.

As mentioned, we observed that successive student questions likely concern concepts that have prerequisite/successor relationships with each other. In our research, we define tutoring sessions to include such questions, and answers to them. 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.

For example, a tutoring session may start with a question about *application program* and ends with an acceptance action. In the session, there may be questions and answers about *application program*, and about its prerequisites like *program*, *programming language*, etc. (according to the DAG in Figure 2). If before the session ends, the student asks a question about *file*, another tutoring session starts.

Tutoring sessions play an important role in our techniques. By dividing a tutoring process into such sessions, we can limit computing in a session to a subset of concepts that have prerequisite/successor relationships with each other. We partition the state space into sub-spaces, and localize the computing in a session within a sub-space. In this way, we reduce $|S|$ involved in making a decision [9]. Based on the partitioned state space, we split the solution space.

B. Structure of Solution Space

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 system at a point in the session, usually for the student to make up some knowledge. We denote a current question by $(?C^c)$, where the 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. A current question may be asked by the student, or made by the system. The original question is also the current question, right after it is asked. In the above example of tutoring session, the question concerning *application program* is the original question. if there is a question about *programming language*, it is a current question.

In a session, the tutoring agent chooses an optimal policy tree from a tree set to answer a question (see (6)). Since the agent's ultimate goal is to teach C^o, and current questions in the session concern prerequisites of C^o, we can include only the policy trees for teaching concepts in $(\wp_{C^o} \cup C^o)$ in the tree set that is evaluated in the session started by $(?C^o)$.

The entire tree set $\mathcal{T}$ can thus be split into subsets of $\mathcal{T}_{C^c}^{C^o}$, where C^o is a concept that can be in an original question and C^c is a concept in $(\wp_{C^o} \cup C^o)$, where $\wp_{C^o}$ includes all direct and indirect prerequisites of C^o. The computing for choosing an action to answer a current question evaluates one subset only: when the original question is $(?C^o)$ and current question is $(?C^c)$, the tree set to evaluate is $\mathcal{T}_{C^c}^{C^o}$. In computing (6) for answering the current question, the $\mathcal{T}$ in the equation is substituted with $\mathcal{T}_{C^c}^{C^o}$.

Tree set $\mathcal{T}_{C^c}^{C^o}$ includes trees for concepts in $(\wp_{C^c} \cup C^c)$, i.e., for C^c and its prerequisites. To answer $(?C^c)$, the agent evaluates all of them, to decide to teach C^c or one of its prerequisites. Let C be a concept in $(\wp_{C^c} \cup C^c)$. In $\mathcal{T}_{C^c}^{C^o}$, there is one or more trees for C. To simplify the discussion here, we assume one tree for C. We denote the tree for C in $\mathcal{T}_{C^c}^{C^o}$ by $\mathcal{T}_{C^c}^{C^o}.\tau_C$. Next, we discuss the structure of $\mathcal{T}_{C^c}^{C^o}.\tau_C$.

C. Structure of a Policy Tree

As discussed, in a tutoring session started by $(?C^o)$, the goal of the agent is to teach C^o. In a policy tree for answering a current question, the leaf nodes must be a student action that accepts $(!C^o)$, which is an action to teach the concept in the original question. 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 include concepts in $\wp_{C^o}$ only. (Student questions are treated as observations.)

The root of $\mathcal{T}_{C^c}^{C^o}.\tau_C$ is $(!C)$, i.e. an action 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 first M children are sub-trees rooted by $(!C_1)$, ... $(!C_M)$ and connected by edges of $(?C_1)$, ... $(?C_M)$. The last child is a sub-tree rooted by $(!C^u)$, where C^u is one of the direct successors of C. This sub-tree is connected by an edge of acceptance action (Θ). For each direct successor of the concept at the root, we construct a tree. The semantics of such root-children structure is that after $(!C)$, if the student asks about a prerequisite of C, teach that prerequisite, if the student accepts $(!C)$, teach one of the direct successors of C.

In a policy tree, each sub-tree is structured in the same way. That is, the root has an edge for each of its prerequisites and an acceptance edge. 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.

Figure 3 illustrates policy tree $\mathcal{T}_{ML}^{ML}.\tau_{ML}$, where ML stands for *machine language*. The prerequisite relationships are based on Figure 2. ML has three direct and indirect prerequisites IN (*instruction*), BD (*binary digit*), and PL (*programming language*). Therefore the root has three edges for the prerequisites and an acceptance edge. Since the root is for answering the original question, the acceptance edge connects to a terminating action (represented by a horizontal bar). Figure 4 shows policy trees $\mathcal{T}_{ML}^{ML}.\tau_{PL}$, $\mathcal{T}_{ML}^{ML}.\tau_{IN}$, and $\mathcal{T}_{ML}^{ML}.\tau_{BD}$. Figure 5 shows policy trees $\mathcal{T}_{PL}^{PL}.\tau_{PL}$ and $\mathcal{T}_{PL}^{PL}.\tau_{IN}$.

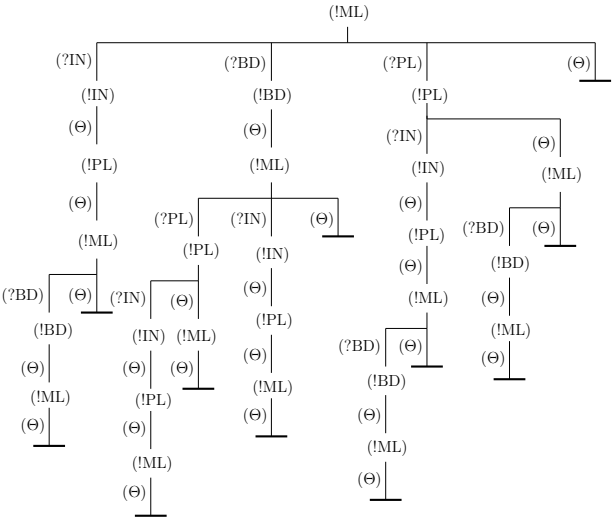

Figure 3. Policy tree $\mathcal{T}_{ML}^{ML}.\tau_{ML}$.

The policy tree structure helps improve efficiency in both space and time. Firstly, it minimizes the number of trees to evaluate. To answer $(?C^c)$ in a session started by $(?C^o)$, the agent evaluates the trees in $\mathcal{T}_{C^c}^{C^o}$ only, instead of all the possible trees. Secondly, it minimizes the costs for evaluating individual trees. A tree deals with only the concepts in $(\wp_{C^o} \cup C^o)$. We can thus minimize the set of observations O. We can also minimize the tree height: a path concerns related concepts only, and unnecessary actions are excluded at the earliest possible point. As discussed before, H and $|O|$ determine the costs for evaluating a tree. Minimizing H and $|O|$ helps minimize tree sizes and thus the space and time costs. In addition, this structure causes no information loss. In a tree, all the possible student actions (observations) have been taken into account.

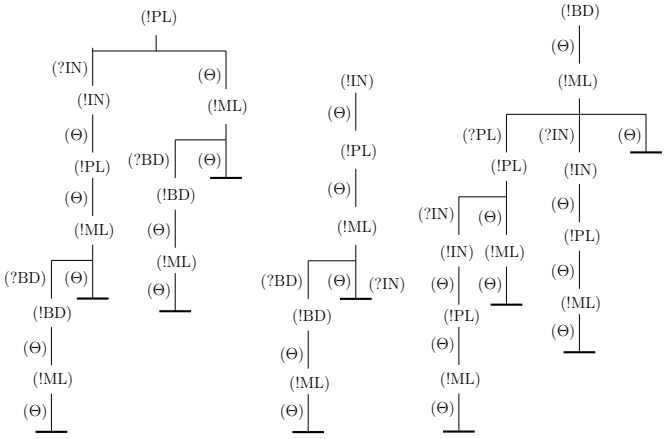

Figure 4. Policy trees $\mathcal{T}_{ML}^{ML}.\tau_{PL}$ (left), $\mathcal{T}_{ML}^{ML}.\tau_{IN}$ (middle), and $\mathcal{T}_{ML}^{ML}.\tau_{BD}$ (right).

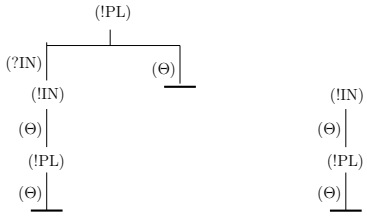

Figure 5. Policy trees $\mathcal{T}_{PL}^{PL}.\tau_{PL}$ (left) and $\mathcal{T}_{PL}^{PL}.\tau_{IN}$ (right).

For better time efficiency, we construct ready-to-use policy trees and store them in a tree database. When the agent needs to make a decision with certain original question and current question, it searches the database for a tree set, and evaluates trees in the set only.

D. Decision-Making with Policy Trees

In a tutoring session started by $(?C^o)$, to answer current question $(?C^c)$, the agent evaluates trees in $\mathcal{T}_{C^c}^{C^o}$ by using (6), and choose the optimal one. In using (6), it substitutes $\mathcal{T}$ with $\mathcal{T}_{C^c}^{C^o}$. For example, when the original and current questions are both (?ML), the agent evaluates the four trees in $\mathcal{T}_{ML}^{ML}$ showed in Figures 3 and 4. It finds the tree of the highest value (optimal tree) based on its current belief.

A policy tree is not a tutoring plan that the agent must follow in the future. It is the choice 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):

1) If the student action is (Θ), and the (Θ) edge connects to a terminating action, the agent terminates the current tutoring session;
2) If the student action is (Θ), and the (Θ) connects to a $(!C)$, the agent considers $(?C)$ as the current question in the next step.
3) 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 rules 2) and 3), 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. Whereas, if after (!ML) the student action is (?PL), the agent considers that (?PL) is the current question in the next step. It evaluates the trees in $\mathcal{T}_{PL}^{ML}$, and continues until it takes a terminating action.

E. Structure for Dealing with Limited Memory

In constructing a tree database, the system creates each policy tree in memory before storing it. To evaluate a policy tree, the system first loads it into memory. Although our technique can minimize the numbers of nodes and edges in individual trees, and we use efficient data structures for tree nodes and edges to reduce memory usage, some trees are still very large. Those are trees for concepts having large numbers of prerequisites. A single tree can be bigger than the memory space available to run the ITS. Dealing with limited memory is a challenging issue in applying a method of policy trees.

The structure we developed for policy trees offers flexibilities in creating and loading policy trees in memory. Let $(?C^o)$ and $(?C^c)$ be original and current questions ($C^c \in (\wp C^o \cup C^o)$). As described, we create tree set $\mathcal{T}_{C^c}^{C^o}$, which includes policy trees for C^c and all its prerequisites. Let C', C", ... be prerequisites of C^c. The trees are $\mathcal{T}_{C^c}^{C^o}.\tau_{C^c}$, $\mathcal{T}_{C^c}^{C^o}.\tau_{C'}$, $\mathcal{T}_{C^c}^{C^o}.\tau_{C"}$, ... According to our rules for structuring policy trees, $\mathcal{T}_{C^c}^{C^o}.\tau_{C'}$, $\mathcal{T}_{C^c}^{C^o}.\tau_{C"}$, ... are subtrees of $\mathcal{T}_{C^c}^{C^o}.\tau_{C^c}$. They are connected to the root by edges of $(?C')$, $(?C")$ ... This structure allows us to physically create $\mathcal{T}_{C^c}^{C^o}.\tau_{C'}$, $\mathcal{T}_{C^c}^{C^o}.\tau_{C"}$, ... only, and create $\mathcal{T}_{C^c}^{C^o}.\tau_{C^c}$ as a root and pointers to the trees. This approach can solve the problem that $\mathcal{T}_{C^c}^{C^o}.\tau_{C^c}$ exhausts the available memory.

For example, when the original and current questions are both (?ML), and ML has prerequisites PL, IN and BD, tree set $\mathcal{T}_{ML}^{ML}$ includes four trees: $\mathcal{T}_{ML}^{ML}.\tau_{ML}$, $\mathcal{T}_{ML}^{ML}.\tau_{PL}$, $\mathcal{T}_{ML}^{ML}.\tau_{IN}$, and $\mathcal{T}_{ML}^{ML}.\tau_{BD}$. The four trees are illustrated in Figures 3 and 4. We can see that $\mathcal{T}_{ML}^{ML}.\tau_{PL}$, $\mathcal{T}_{ML}^{ML}.\tau_{IN}$, and $\mathcal{T}_{ML}^{ML}.\tau_{BD}$. are the first three subtrees of $\mathcal{T}_{ML}^{ML}.\tau_{ML}$. We can physically create $\mathcal{T}_{ML}^{ML}.\tau_{PL}$, $\mathcal{T}_{ML}^{ML}.\tau_{IN}$, and $\mathcal{T}_{ML}^{ML}.\tau_{BD}$. For $\mathcal{T}_{ML}^{ML}.\tau_{ML}$, we include root (!ML) and edges (?IN), (?BD), and (?PL) only.

Another structural feature of the policy trees can be used to save storage space. When C^{o_1} is a prerequisite of C^{o_2}, The trees in $\mathcal{T}_{C}^{C^{o_2}}$ can be used in $\mathcal{T}_{C}^{C^{o_1}}$ with minor changes. Take $\mathcal{T}_{ML}^{ML}.\tau_{PL}$ and $\mathcal{T}_{PL}^{PL}.\tau_{PL}$ (illustrated in Figures 4 and 5) as an example. The two trees are both for PL but in different tree sets ($\mathcal{T}_{ML}^{ML}$ and $\mathcal{T}_{PL}^{PL}$). PL is a prerequisite of ML. By substituting (!ML) in $\mathcal{T}_{ML}^{ML}.\tau_{PL}$ with a terminating action (represented as a horizontal bar), the tree can be used as $\mathcal{T}_{PL}^{PL}.\tau_{PL}$. This allows us to create and store a tree in a tree set and use it in multiple sets with minor changes. This structure may help save storage space for the tree database.

VII. EXPERIMENTS

We experimented our system using a data set of software basics. This data set contains 90 concepts. A concept has zero to five prerequisites. In the following, we first present the results concerning system performance in adaptive tutoring, and then the results concerning space usage.

30 students participated in the experiments, randomly divided into two groups of the same size. Each student studied with the ITS for about 45 minutes. The students were adults who knew how to use desktop or laptop computers, or smart phones, and application programs, including Web browsers, email systems, text processors, and phone apps. None of the students took a course on computer software before the experiments. The ITS taught students in the first group with the POMDP turned off. When a student asked about a concept, the system either taught the concept directly, or randomly selected a prerequisite to teach. The ITS taught students in the second group with the POMDP turned on. The system chose the optimal action when answering a question.

The performance perimeter was *rejection rate*. Roughly, if right after the system taught concept C, the student asked a question about a prerequisite of C, or said "I already know C", we considered the student rejected the system action. For a student, the rejection rate is calculated as the ratio of the number of system actions rejected by the student to the total number of system actions for teaching the student. The rejection rate of a student could be used to measure how the student was satisfied with the teaching, and thus measure the system's abilities to choose optimal actions.

TABLE I. 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	$s_1^2 = 0.0158$	$s_1^2 = 0.0113$

We applied a two-sample *t*-test method to evaluate the effects of the optimized teaching strategy to the teaching performance of an ITS. For the two groups, we calculated means $\bar{X}_1$ and $\bar{X}_2$. Sample mean $\bar{X}_1$ was used to represent population mean μ_1, and $\bar{X}_2$ represent μ_2. The alternative and null hypotheses were:

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

The means and variances calculated for the two groups are listed in Table I. The mean rejection rate in Group 1 was 0.5966 and the mean rejection rate in Group 2 was 0.2284. The statistical analysis suggested we could reject H_0 and accept H_a. That is, the difference between the two means was significant.

In the following, we discuss the results related to space usage. As described, we partitioned the state space, so that we

TABLE II. NUMBERS OF CONCEPTS, TREE SETS, TREES, AND TREE HEIGHTS IN SUB-SPACES.

Sub-space	# of concepts	# of tree sets	# of trees	Max set size	Max inheight
1	21	98	285	18	30
2	23	134	456	25	22
3	20	111	388	22	26
4	27	188	753	29	37
5	25	173	684	26	32
6	26	169	682	31	35

could localize computing in a tutoring session within a sub-space. For each sub-space, we created tree sets, each of which contained policy trees to be evaluated for certain questions. For the data set of software basics, our algorithms partitioned the state space into six sub-spaces.

Table II lists the numbers of concepts, tree sets, and trees in each sub-space. It also lists the maximum size of tree sets and maximum tree height in each sub-space. It can be seen that the largest tree set contained 31 trees, and the maximum tree height was 37. That is, in the worst case, to answer a question the system evaluated a set of 31 trees of maximum height 37. Such tree sets and heights did not create major efficiency problems for a modern computer. When the experimental ITS run 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.

TABLE III. MEMORY CONSUMPTION OF ITS COMPONENTS.

ITS Component	Storage usage (MB)
States	3
$P(s'\|s,a)$	5,319
$P(o\|a,s')$	44
$R(s,a)$	13
Tree database	3,671
Policy tree values	37

Table III includes the information about storage usage. The ITS components consuming memory/disk space are states, state transition probabilities $P(s'|s,a)$, observation probabilities $P(o|a,s')$, expected rewards $R(s,a)$, and policy trees. We also saved tree values (computed by using (3)) for possible re-use, for better response time. The tree values were re-computed when the probabilities were updated. The tree database consumed 3,671 MB. Our techniques for structuring policy trees helped reduce the tree database to a manageable size. Before the space-saving structures were used, the tree database consumed about three times the space, and some policy trees could not be created because the available memory was exhausted.

We also experimented the techniques with a data set of statistics. This data set included all the 231 concepts taught in a textbook on introductory statistics [5]. The state space was partitioned into 28 subspaces. The techniques were effective in dealing with space efficiency issues with this subject. In this paper, we will not present the experimental results because of the limited page space.

VIII. CONCLUSION

The issue of space efficiency has been a major obstacle in building a POMDP-based intelligent tutoring system. Policy trees may consume very large space, even exhaust the available memory to crash a system. We developed a set of techniques to address the space problems caused by policy trees. With the techniques, we could minimize the number of trees, and minimize the sizes of individual trees. We could further reduce the space consumption by allowing trees to share the same tree components. Encouraging results have been achieved in experiments.

ACKNOWLEDGMENT

This research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC). Patrick Hartman implemented part of the system and conducted some of the experiments.

REFERENCES

[1] A. Cassandra, "A survey of pomdp applications", *Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Process*, Oct 23-25, 1998, Orlando, FL, USA. AAAI Press, Palo Alto, CA, USA, July, 1998, pp. 17-24.

[2] B. Cheung, L. Hui, J. Zhang, and S. M. Yiu, "SmartTutor: an intelligent tutoring system in web-based adult education", *The journal of Systems and software*, Elsevier, Cambridge, MA, USA, vol. 68, pp. 11-25, 2003, ISSN: 0164-1212.

[3] H. R. Chinaei, B. Chaib-draa, and L. Lamontagne, "Learning Observation Models for Dialogue POMDPs", in *Canadian AI'12 Proceedings of the 25th Canadian conference on Advances in Artificial Intelligence*, May 28-30, 2012, Toronto, ON, Canada, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 280-286, ISBN: 978-3-642-30353-1.

[4] J. T. Folsom-Kovarik, G. Sukthankar, and S. Schatz, "Tractable POMDP Representations for Intelligent Tutoring Systems", *ACM Transactions on Intelligent Systems and Technology*, New York, NY, USA vol. 4, pp. 29:1-29:22, 2013, ISSN: 2157-6904.

[5] G. W. Heiman, *Basic Statistics for the Behavioral Sciences, Sixth Edition*, WadSworth, Cengage Learning, Belmont, CA, 2011, ISBN-13: 978-0-8400-3143-3.

[6] A. N. Rafferty, E. Brunskill, L. Thomas, T. J. Griffiths, and P. Shafto, "Faster Teaching by POMDP Planning", in *Proceedings of Artificial Intelligence in Education (AIED) 2011*, June 28 - July 2, 2011, Auckland, New Zealand. Springer, New York, NY, USA, July, 2011, pp. 280-287, ISBN: 078-3-642-21869-9.

[7] G. Theocharous, R. Beckwith, N. Butko, and M. Philipose, "Tractable POMDP Planning Algorithms for Optimal Teaching in SPAIS", in *IJCAI PAIR Workshop (2009)*, July 11-17, 2009, Pasadena, CA, USA. AAAI Press, Palo Alto, CA, USA, July, 2009, ISBN: 978-1-57735-426-0.

[8] K. VanLehn, B. van de Sande, R. Shelby, and S. Gershman, "The Andes Physics Tutoring System: an Experiment in Freedom", in Nkambou et-al eds. *Advances in Intelligent Tutoring Systems*, Berlin Heidelberg: Springer-Verlag, 2010, pp. 421-443, ISBN: 3642143628.

[9] F. Wang, "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)*, Okayama, Japan, July, 2015, pp. 67-72, ISBN: 978-1-4799-9957-6.

[10] F. Wang. "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)*, Rome, Italy, April, 2016, pp. 85-93, ISBN: 978-989-758-179-3.

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

[12] J. D. Williams, and S. Young, "Partially observable Markov decision processes for spoken dialog systems", *Computer Speech and Language*, Elsevier, Cambridge, MA, USA, vol. 21, pp. 393-422. 2007, ISSN: 0885-2308.

[13] B. P. Woolf, *Building Intelligent Interactive Tutors*, Burlington, MA, USA: Morgan Kaufmann Publishers, 2009, ISBN 978-0-12-373594-2.

 ISBN: 978-1-61208-576-0