Empirically Evaluating the Application of Reinforcement Learning to the Induction of Effective and Adaptive Pedagogical Strategies

Min Chi (minchi@cs.cmu.edu)
Machine Learning Department, Carnegie Mellon University, PA USA

Kurt VanLehn (kurt.vanlehn@asu.edu)
School of Computing, Informatics and Decision Science Engineering, Arizona State University, AZ USA

Diane Litman (litman@cs.pitt.edu)
Department of Computer Science & Intelligent Systems Program & Learning Research and Development Center, University of Pittsburgh, PA USA

Pamela Jordan (pjordan@pitt.edu)
Learning Research and Development Center, University of Pittsburgh, Pittsburgh, PA USA

November, 2010

Abstract. For many forms of e-learning environments, the system's behavior can be viewed as a sequential decision process wherein, at each discrete step, the system is responsible for selecting the next action to take. Pedagogical strategies are policies to decide the next system action when there are multiple ones available. In this project we present a Reinforcement Learning (RL) approach for inducing effective pedagogical strategies and empirical evaluations of the induced strategies. This paper addresses the technical challenges in applying RL to Cordillera, a Natural Language Tutoring System teaching students introductory college physics. The algorithm chosen for this project is a model-based RL approach, Policy Iteration, and the training corpus for the RL approach is an exploratory corpus, which was collected by letting the system make random decisions when interacting with real students. Overall, our results show that by using a rather small training corpus, the RL-induced strategies indeed measurably improved the effectiveness of Cordillera in that the RL-induced policies improved students' learning gains significantly.

Keywords: reinforcement learning, pedagogical strategy, machine learning, human learning

1. Introduction

For many forms of e-learning environments, the system's behaviors can be viewed as a sequential decision process wherein, at each discrete step, the system is responsible for selecting the next action to take. Pedagogical strategies are defined as policies to decide the next system action when there are multiple ones available. Each of these system decisions affects the user's successive actions and performance. Its impact on student learning cannot often be observed immediately and the effectiveness of one decision also depends on the effectiveness of subsequent decisions. Ideally, an effective learning environment should craft and adapt its decisions to users' needs [4, 52]. However, there is no existing well-established theory on how to make these system decisions effectively.

In this paper, we focus on one form of highly interactive e-learning environment, Intelligent Tutoring systems (ITSs). Most existing ITSs either employ fixed pedagogical policies providing little adaptability or employ hand-coded pedagogical rules that seek to implement existing cognitive or instructional theories [4, 41, 66]. These theories may or may not have been well-evaluated. For example, in both the CTAT [4, 41] and Andes systems [69], help is provided upon request because it is assumed that students know when they need help and will only process help when they desire it. Research on gaming the system, however, has raised some doubts about this. It showed that students sometimes exploit these mechanisms for shallow gains thus voiding the help value [6, 5].

Previous researchers have largely treated the specification of pedagogical policies as a design problem: several versions of a system are created, the only difference among them being the pedagogical policies employed [47, 20, 56]. Data is then collected from human subjects interacting with each version of the system, and the students' performance is then statistically compared. Due to cost limitations, typically, only a handful of alternative policies are explored. Yet, many such other reasonable ones are still possible.

In recent years, work on the design of ITSs and Natural Language (NL) dialogue systems has involved an increasing number of data-driven methodologies. Among these, Reinforcement Learning (RL) has been widely applied [8, 30, 31, 32, 59, 46, 62, 72, 71]. In the project reported here, we applied a form of RL, Policy Iteration, to improve the effectiveness of an ITS by inducing pedagogical policies directly from an exploratory corpus. The exploratory corpus was collected by letting the ITS make random decisions when interacting with real students.

Compared with previous studies on applying RL to induce pedagogical policies on ITSs, this project makes at least three major contributions. First, we show that using a relatively small exploratory training corpus is a feasible approach to induce effective pedagogical policies. Second, we empirically show that the RL induced policies indeed improved students' learning gains significantly. Third, while much of the previous research on applying RL to ITSs and non-tutoring NL dialogue systems used pre-defined state representation, our approach in this project is to begin with a large set of features to which a series of feature-selection methods were applied to reduce them to a tractable subset. We will shed some light on the relative effectiveness of different feature-selection methods and which features among the ones defined were most involved in the final induced policies.

Overall, our results provide empirical evidence that, when properly applied, RL can be applied to reach rather complicated and important instructional goals such as learning and quantitatively and substantially improve the performance of a tutoring system. In this paper, we will describe our detailed methodology for using RL to optimize the pedagogical policies based on limited interactions with human students and then present empirical results from validating the induced policies on real students.

In the following, section 2 gives a brief overview of prior related research on applying RL to either ITS or NL dialogue systems. Section 3 explains how RL can be used to optimize pedagogical policies with a set of given features, section 4 explains our procedure on how a series of feature-selection methods were applied to a large set of features in this project in order to induce the "best" policy. Section 5 describes our general approach and section 6 describes the general methods including an introduction of the Cordillera system and procedures, while section 7 describes how Cordillera optimizes its decisions from experimentally obtained dialogue data. Section 8 reports empirical results evaluating the performance of Cordillera's induced pedagogical strategies and demonstrates that the effectiveness of Cordillera was improved.

2. Background

Generally speaking, a RL model consists of three elements: 1. $S = \{S_1, \ldots, S_n\}$ is a state space; 2. $A = \{A_1, \ldots, A_m\}$ is an action space represented by a set of action variables; 3. $R = r(s_i, s_j, a_k)$ denotes a reward model that assigns rewards to state transitions. The goal of RL is to find an optimal policy π^* that maps each state to the proper action that would generate the maximum rewards.

The empirical procedure of applying RL to either ITS or non-tutoring NL dialogue systems can be divided into two phases: a training phase and a test phase. The training phase mainly involves defining an appropriate state representation S, a reasonable action space A, and an appropriate reward function R, collecting a training corpus Γ, and applying some RL algorithms to induce policies. In order to apply RL to induce an effective policy, it is important for the system to explore the relevant S of possible decision action sequences during the training phase. A common problem in RL is finding a balance between exploration (attempting to discover more about the world) and exploitation (using what we already know about the world to get the best results we can). On one hand, if the system does not explore enough, the RL approach might not find an effective policy at all. On the other hand, if the system explores too much, it cannot stick to a path; in fact, it is not really learning as it cannot exploit its knowledge, and so acts as though it knows nothing. It is often unclear how much exploration should be done in order to induce an effective policy. The test phase generally involves some sorts of evaluations to find out whether the RL-induced policies indeed fulfill their promises.

There are two major categories of RL, model-free algorithms and model-based algorithms [38]. Model-free algorithms learn a value function or policy directly from the experience while interacting with the agent; at the end of learning, the agent knows how to act, but does not explicitly know anything about the environment. Whereas model-based algorithms first construct a model of the state transition and outcome structure of the environment, and then evaluate actions by searching this model. In other words, Model-based methods do explicitly learn state transitions. Generally speaking, model-free methods are appropriate for domains where collecting data is not a big issue; their corresponding algorithms include Monte Carlo methods and temporal difference methods. Model-based methods, on the other hand, are appropriate for domains where collecting data is expensive; their corresponding algorithms include dynamic programming such as Policy Iteration and Value Iteration.

Previously both model-free and model-based RL algorithms were applied to improve conventional ITSs. Next, we will briefly overview related previous research.

2.1. Previous Research On Applying RL to ITSs

Beck et al. [8] investigated applying RL to induce pedagogical policies that would minimize the time students take to complete each problem on AnimalWatch, an ITS that teaches arithmetic to grade school students. In the training phase of their study, they used simulated students. Given that the cost of collecting data with simulated students is relatively low, a model-free RL method, Temporal Difference learning, was applied. In the test phase, the induced policies were added to AnimalWatch and the new system was empirically compared with the original version of AnimalWatch. Results showed that the policy group spent significantly less time per problem than their no-policy peers. Note that their pedagogical goal was to reduce the amount of time per problem, however faster learning does not always result in better learning performance. Nonetheless, their results showed that RL can be successfully applied to induce pedagogical policies for ITSs.

Iglesias and her colleagues [30, 31, 32], on the other hand, focused on applying RL to improve the effectiveness of an Intelligent Educational System that teaches students DataBase Design. They applied another model-free RL algorithm, Q-learning. The goal for the induced policy was to provide students with direct navigation support through the system's content with the expectation that this would help students learn more efficiently. In the training phase, similar to Beck et al.'s approach, Iglesias and her colleagues used simulated students. In the test phase, the induced policy was also empirically evaluated on real students. Results showed that while the policy led to more effective system usage behaviors from students, the policy students did not outperform the no-policy peers in terms of learning outcomes.

Martin and Arroyo [46] applied a model-based RL method, Policy Iteration, to induce pedagogical policies that would increase the efficiency of hint sequencing on the Wayang Outpost web-based ITS. During the training phase, the authors used a student model to generate the training data for inducing the policies. Here the student model was similar to the simulated students used in Beck et al and Iglesias et al. In the test phase, the induced policies were tested on the student model again and results showed that the induced policies increased its predicted learning. However, since the policies were not tested with human students, their effectiveness with real students is still an open question.

Tetreault et al [62, 64] used an Intelligent Spoken Tutoring System, ITSPOKE, which teaches students college physics [44]. In their work, they used a previously collected corpus of physics tutorial dialogues as a training corpus and investigated applying Policy Iteration to induce pedagogical policies from it. The focus of their work was introducing a novel method for evaluating state representations and thus the learning gains of human students using the induced policy system were not measured and compared to the prior system. Additionally, note that because the training corpus used in this work was not collected with the goal of exploring the full range of tutorial decisions, the tutor often executed only one type of action in many dialogue states.

Although there have been other studies on application of RL to ITSs, they mostly involved inducing domain models rather than pedagogical policies. For example, Barnes and Stamper [7, 60] have applied RL to construct problem solutions from existing students' solutions for an ITS called Proofs Tutorial, which teaches college-level discrete mathematics. They used a form of the model-based RL method Value Iteration and the resulting problem solutions were then used to generate hints for new students. They found that the extracted solutions and the proposed hint-generating functions were able to provide hints over 80% of the time for the new students.

In short, both model-free and model-based RL algorithms have been applied to induce pedagogical policies in ITSs but only model-free RL induced policies were empirically evaluated on real students [8, 30, 31, 32]. As far as we know, none of the previous research has empirically shown that an RL induced policy significantly improved students' learning performance. Given that improving learning is a primary goal for any ITS, a better RL approach together with empirical evaluations is needed and is attempted in this project.

Note that previous research on applying RL to induce pedagogical policies either used pre-existing data that was collected for other purposes [62, 64] or simulated students or models [8, 30, 31, 32, 46, 1] during the training phase. We argue that neither approach is optimal.

Using existing datasets is complicated by the fact that pre-existing systems often explore a small part of the state-action space and thus may yield biased or limited information. When using simulated students or models, however, the effectiveness of the induced policies would largely depend on the accuracy of the simulated students. Building accurate simulated students is not easy and it is especially challenging because human learning is a rather complex, poorly understood process. For instance, one possible explanation for the difference between the policy and no-policy groups that was found in [8] but not in Iglesias et al's work [30, 31, 32] may be that while it is possible to accurately model time on task in [8], modeling how students would respond to the system and simulating how students would learn is much more challenging in [30, 31, 32].

On the other hand, previous research on applying RL to improve dialogue systems employed an alternative approach that involved collecting an exploratory corpus from human subjects in the training phase. Next, we will briefly describe how previous research in dialogue systems tackled the issue.

2.2. Collecting An Exploratory Corpus

Dialogue Systems are a field of Computer Science that focuses on the construction of computer systems that interact with human users via natural-language dialogues. Much of the work in this area is focused on systems that obtain information or search databases such as querying bus schedules [54] and booking airline tickets [57]. In recent years, RL has been widely applied to the design of dialogue systems [72, 71, 59].

In [59], Singh et al. explored avoiding biased training data by collecting an exploratory corpus. In their study, they used a dialogue system called NJFun, a real-time spoken dialogue system that provides users with information about things to do in New Jersey. In order to let the system explore the relevant space of possible decision sequences, they collected an exploratory corpus from a system that makes random system decisions as it spoke with human users, thus ensuring that the transitions are adequately explored [59]. Then a model-based RL method, Value Iteration [58], was applied to induce an effective dialogue policy from the collected exploratory training corpus. In the test phase, they empirically evaluated the induced policy on real users and showed that the dialogue policies significantly improve the likelihood of task completion on NJFun. Similar approaches have been shown to be successful in other studies [72, 71].

Therefore, in this project we use neither pre-existing system-user interaction data nor simulated students and instead followed the approach of Singh et al [59]. More specifically, in the training phase, we collected an exploratory corpus by training human students on an ITS that makes random decisions and then applied RL to induce pedagogical policies from the corpus; and in the test phase, we re-implemented the ITS using the learned pedagogical policies and measured the learning gains of the induced policies with a new group of students. The ITS involved in this project is called Cordillera, a NL tutoring system which teaches students introductory physics [68].

Although NL tutoring systems can be seen as a type of dialogue system, it is an open question whether using an exploratory corpus as training data would induce policies that improve learning gains as opposed to task completion or user satisfaction.

One major source of uncertainty comes from the fact that the rewards used in RL are much more delayed in NL tutoring systems than in non-tutoring dialogue systems. The most preferable rewards for NL tutoring systems are student learning gains and the rewards for non-tutoring dialogue systems are often user satisfaction or task completion. Even though the rewards in both types of systems will not be available until the entire system-user interaction is over, NL tutoring dialogues are longer and more complex than the database-access dialogues described above.

In dialogue systems like a train scheduler or NJFun, the interaction time is often less than 20 minutes and the number of user-system interactions is generally less than 20 turns [58, 59]. In NL tutoring systems, on the other hand, the preparatory training materials and tests typically exceed these timeframes significantly. In this project, it took students roughly 4-9 hours to complete the training on Cordillera and the number of Cordillera-student interactions was more than 280.

More immediate rewards are more effective than more delayed rewards for RL-based policy induction. This is because the issue of assigning credit for a decision or attributing responsibility to the relevant decision is substantially easier with more immediate rewards. The more we delay success measures from a series of sequential decisions, the more difficult it becomes to identify which of the decision(s) in the sequence are responsible for our final success or failure.

Another major source of uncertainty for using an exploratory corpus to induce effective pedagogical policies comes from the fact that the amount of training data is severely limited by the need for student-ITS interactions. Compared with non-tutoring dialogue systems, collecting data on NL tutoring systems is even more expensive. In this study, we only included 64 students' interaction logs in the exploratory corpus which is quite small compared with 311 dialogues used in NJFun [59] whose domain task is much simpler than ours.

To summarize, given the much delayed reward functions and the high cost of collecting data, it is still an open question whether applying Policy Iteration to induce pedagogical policies directly from a relatively small exploratory training corpus would indeed be effective in this project. Moreover, while previous research on applying RL to ITSs and dialogue systems gave us some guidance on the type of RL algorithms we should apply and how to collect the training corpus, we still face one primary challenge: how to define the state representation.

2.3. Prior Work on State Representation

For RL, as with all machine learning tasks, success depends upon an effective state representation S. Ideally S should include all of the relevant dialogue history necessary to determine which action should be taken next. One obvious but impractical choice is to use a complete record of the dialogue to the present point; however, in practice we need to compress the dialogue history to make the space tractable. In other words, an effective representation S should be an accurate and compact model of the learning context. The challenge thus lies in identifying useful state features.

Early work on applying RL on non-tutoring dialogue systems focused largely on relatively simple task domains and used expert defined slot-based state representations. In applying RL to improve NJFun, for example, seven features were included in their representation S which includes information such as the confidence of ASR (automatic speech recognition), whether the system has greeted the user, which information is being worked on (time, actitivity, location), and how many times a given piece of information has been asked for [58]. Additionally, some previous studies focused on the type of features that should be included in the state representation. For instance, Frampton and colleagues [24, 25] showed that incrementally adding high-level contextual information (such as the user's last dialogue act and the last system move) into a state representation was beneficial.

As RL and MDP are applied to more complex domains, S may increase rapidly in size and complexity. For example, when a student is trained on an ITS, there are many factors that might determine whether the student learns well from the ITS. Compared with non-tutoring NL dialogue systems, where success is primarily a function of communication efficiency, communication efficiency is only one of the factors determining whether a student learns well from an ITS. Moreover, many other factors are not well understood, so to be conservative, states need to contain features for anything that is likely to affect learning.

To make the RL problem tractable, much prior work on applying RL to ITSs also used pre-defined state representation. More specifically, the representation S often consists of features that were suggested by the learning literature. In the work of Iglesias and her colleagues' [30, 31, 32], the state representation involves the student's knowledge level on various knowledge items. By doing so, they assumed that the system should adapt its behavior based on student knowledge levels. However, much other useful information about the learning environment, such as the difficulty of the problem, student motivation and affect, was largely ignored. By contrast, Tetreault et al. included a broader range of features in their state representation: *Certainty*, *Correctness*, *Percent Correct*, *Concept Repetition*, and *Frustration* [62, 64]. Note that some of their features were manually annotated. Beck et al. [8] included 15 state features from 4 main categories that are suggested by the learning literature. The four main categories included the student's level of prior proficiency, level of cognitive development, and background, and the difficulty of the current problem [8].

While existing learning literatures and theories give helpful guidance on state representation S, we argue that such guidance is often considerably more general than the specific state features chosen. The following, for example, is a list of six features describing a student's knowledge level from different perspectives.

1. [**Percentage Correct:**] Defined as the number of the correct student entries divided by the total number of the student entries.

2. [**Number of Correct:**] Defined as the number of the correct student entries.

3. [**Percent Correct in This Session:**] Defined as the number of the correct student entries in this session divided by the total number of student entries in this session.

4. [**Number of Correct in This Session:**] Defined as the absolute number of the correct student entries in this session.

5. [**Number of Incorrect:**] Defined as the number of the incorrect student entries.

6. [**Number of Incorrect in This Session:**] Defined as the number of the incorrect student entries in this session.

When making specific decisions about including a feature on student knowledge level in S, for example, it is often not clear which one of these six features should be included. Therefore a more general state representation approach is needed. To this end this project began with a large set of features to which a series of feature-selection methods were applied to reduce them to a tractable subset. While much previous research on the use of RL to improve ITSs and dialogue systems has focused on developing the best policy for a set of given features [8, 30, 31, 32, 71, 29], only a little work has been done on feature selection.

Paek and Chickering's work, for example, showed how a state-space can be reduced by only selecting features that are parents of the local immediate reward in that the former performs just as well as a more complicated model with other non-parent variables [50]. In [55], Rieser and Lemon used logistic regression to select the best state features for a multi-modal dialogue system and showed marked improvement over the baseline and some supervised learning methods. Most recently, Tetreault, et al [65] tackled the feature selection issue by exploring three evaluation metrics for assessing the utility of adding a particular state feature to a model of user state. The feature selection procedure employed in this project is motivated by their work [65]. However, our approach is fundamentally different from Tetrault et al.'s work because they explored three evaluation metrics and used a relatively simple feature selection procedure. Here we explore a series of different feature selection procedures, but use only one evaluation metric, the Expected Cumulative Reward (ECR).

To summarize, while previous work on applying RL to induce pedagogical policies in ITSs indicated some potential benefits from RL-induced policies, little empirical evidence has shown that an RL induced policy indeed improved students' learning significantly. In this project, to tackle this problem directly, our approach was: 1) to involve human subjects in both training and test phases; 2) to begin with a large set of features to which a series of feature-selection methods were applied to reduce them to a tractable subset.

Previously, we reported our first round of RL results in [15]. Overall, our results showed that even though the RL-induced policies generated significantly different patterns of tutorial decisions than the random policy, no significant difference was found between the two groups on overall learning performance [15]. Only on one specific skill and one particular post-test measurement did the induced policy students score higher than the random policy students [15].

There are many possible explanations for the lack of improvement in our first round of RL policy induction. We argue it was because our RL approach was limited. More specifically, a greedy-like feature selection approach selected four features out of a total of 18 feature choices. In [14], we explored four RL-based feature selection methods on the 18 features. We found that simply improving the feature selection methods resulted in policies with much higher ECR than those employed in [15]. The higher the ECR value of a policy, the better the policy is supposed to perform. However, the effectiveness of induced policies from [14] were not tested with human students.

We then executed a second round of policy induction. As reported earlier [16], the empirical results showed that the second round induced policies indeed caused students to learn more and deeper. To investigate why this welcome benefit occurred, for this paper we analyzed computer logs to characterize the induced policies. Given the page limitations in the previous publications, a complete and detailed description of our RL methodology has not been reported previously.

In this paper, we will present our RL methodology, empirical results from validating the induced policies, log analysis on the characteristics of the induced policies, findings on which features among the ones defined were most involved in the final induced policies, and the relative effectiveness of different feature-selection methods. More specifically, we will mainly focus on the second round of RL policy induction because it turned out to be more successful.

3. Applying RL to Induce Policies with a Set of Given State Features

Previous research on using RL to improve dialogue systems (e.g. [43, 58]) has typically used MDPs (Markov decision process) [61] to model dialogue data. An MDP describes a stochastic control process and formally corresponds to a 4-tuple (S, A, T, R), in which:

$S = \{S_1, \cdots, S_n\}$ is a state space.

$A = \{A_1, \cdots, A_m\}$ is an action space represented by a set of action variables;

$T : S \times A \times S \rightarrow$ ***[0, 1]*** is a set of transition probabilities between states that describe the dynamics of the modeled system; for example: $P(S_j|S_i, A_k)$ is the probability that the model would transition from state S_i to state S_j by taking action A_k.

$R : S \times A \times S \rightarrow R$ denotes a reward model that assigns rewards to state transitions and models payoffs associated with such transitions.

Additionally, $\pi : S \rightarrow A$ is defined as a policy.

From the RL and MDP perspective, pedagogical strategies are simply a set of *policies* which are mappings from the set of states in the state space S to a set of actions in A. Note that in order for RL to be feasible, the number of states and actions should not be too large. On the other hand, when an MDP is constructed for tutorial dialogues, it can have millions of distinct dialogue states and tutor utterances. Here both states and actions in an MDP are abstract. For instance, a state in the MDP might be a set of features that represent thousands of real dialogue states and the action "Elicit" in the MDP might denote any information-seeking questions asked by the tutor. Thus, "state" and "action" have different meanings in an MDP versus in a tutorial dialogue. In order to induce a policy from the MDP perspective, there must be deterministic functions for mapping from dialogue states to MDP states and from a dialogue action to an MDP action. For instance, one type of tutorial decision we investigated is *elicit/tell (ET) decisions*; a policy would determine, based on the features present in a particular dialogue state, whether the tutor should *elicit* the next step from students, or *tell* students the next step directly.

In this project, we applied Policy Iteration [61], which is a model-based RL approach. One major difference between the model-based and model-free RL methods is that the former need to learn transition probabilities T explicitly. Generally T is estimated from a training corpus Γ.

As each student solves a series of training problems on an ITS, a system-student tutorial dialogue is generated. For each tutorial dialogue, a scalar performance measure called reward, R, is calculated. For example, a common choice for R in ITSs is student learning gains. In this project, the reward function R is based on a Normalized Learning

Gain (NLG). This is because NLG measures a student's gain *irrespective of his/her incoming competence* and we have: $NLG = \frac{posttest-pretest}{1-pretest}$. Here *posttest* and *pretest* refer to the students' test scores before and after the training respectively; and 1 is the maximum score.

After training a group of students on the ITS, we get a training corpus Γ, which is a collection of system-student tutorial dialogues. Following Singh et al. (1999), we can view each system-student interaction log d_i as a trajectory in the chosen state space determined by the system actions and student responses:

$$s^1_{d_i} \xrightarrow{a^1_{d_i}, r^1_{d_i}} s^2_{d_i} \xrightarrow{a^2_{d_i}, r^2_{d_i}} \cdots s^{n_{d_i}}_{d_i} \xrightarrow{a^{n_{d_i}}_{d_i}, r^{n_{d_i}}_{d_i}}$$

Here $s^j_{d_i} \xrightarrow{a^j_{d_i}, r^j_{d_i}} s^{j+1}_{d_i}$ indicates that at the j_{th} turn in the tutorial dialogue d_i, the system is in MDP state $s^j_{d_i}$, executes MDP action $a^j_{d_i}$, receives MDP reward $r^j_{d_i}$, and then transfers into MDP state $s^{j+1}_{d_i}$. The number of turns in d_i is n_{d_i}. In this project, only terminal dialogue states have non-zero rewards because a student's learning gain will not be available until the entire tutorial dialogue is completed.

Dialogue sequences obtained from the training corpus Γ then can be used to empirically estimate the transition probabilities T as: $T = \{p(S_j|S_i, A_k)\}^{k=1,\cdots,m}_{i,j=1,\cdots,n}$. More specifically, $p(S_j|S_i, A_k)$ is calculated by taking the number of times that the dialogue is in MDP state S_i, the tutor took action A_k, and the dialogue was next in state S_j divided by the number of times the dialogue was in S_i and the tutor took A_k. The reliability of these estimates depends upon the size and structure of the training data.

Once an MDP model has been completed, calculation of an optimal policy is straightforward. This project employed an RL toolkit developed by Tetreault and Litman [65].

3.1. Tetreault and Litman's Rl Toolkit

Tetreault, & Litman's toolkit [65, 63, 64] uses a dynamic programming algorithm for Policy Iteration [61]. The code was originally built on the MDP toolkit written in Matlab [11]. The purpose of this algorithm is to handle the problem of reward propagation. As noted above, rewards, in this case learning gains, are not assigned until the end of the tutoring process, long after most actions have occurred. The dynamic programming algorithm propagates the rewards back to the internal states weighting the V-value of each state, s, via the following recursive equation:

$$V(s) = \max_{a'} R(s,a) + \sum_{s'} P(s'|s,a)\gamma V(s') \tag{1}$$

Here $P(s'|s,a)$ is the estimated transition model T, $R(s,a)$ is the estimated reward model, and $0 \leq \gamma \leq 1$ is a discount factor. If γ is less than 1, then it will discount rewards obtained later. For all the studies reported here, a discount factor of 0.9 was used, which is common in other RL models [65].

The V-values, as defined by Equation 1, can be estimated to within a desired threshold using policy iteration [61]. Here an estimated V-value and a best possible action to take for each state are recorded. These are then iteratively updated based on the values of its neighboring states. This iteration stops when each update yields a difference below some threshold ϵ. Once the policy iteration process is complete, the optimal dialogue policy $\pi*$ is obtained by selecting the action that produces the highest expected reward (or V-value) for each state.

Besides inducing an optimal policy, Tetreault, & Litman's toolkit also calculate the Expected Cumulative Reward (ECR) and a 95% confidence interval for the ECR (hereafter, 95%CI) for the optimal policy [65]. The Expected Cumulative Reward (ECR) of a policy is derived from a side calculation in the policy iteration algorithm: the V-values of each state, the expected reward of starting from that state and finishing at one of the final states. More specifically, the ECR of a policy π can be calculated as follows:

$$ECR_{\pi} = \sum_{i=1}^{n} \frac{N_i}{N_1 + \cdots + N_n} \times V(s_i) \tag{2}$$

Where $s_1, \cdots, s_n$ is the set of all starting states and $v(s_i)$ is the V-values for state s_i; N_i is the number of times that s_i appears as a start state in the model and it is normalized by dividing $\frac{N_i}{N_1+\cdots+N_n}$. In other words, the ECR of a policy π is calculated by summing over all the initial start states in the model space and weighting them by the frequency with which each state appears as a start state. The higher the ECR value of a policy, the better the policy is supposed to perform.

Tetreault and Litman pointed out one limitation of using the ECR as an evaluation metric for a policy: it assumes that there is sufficient collected data to derive a reliable policy [65]. However, in practice researchers frequently have to deal with issues of data sparsity. They proposed a novel approach of taking into account the reliability of the transition probability T and constructing a confidence interval for the ECR for the learned policy.

As described earlier, the transition probabilities T were derived from the training corpus. Note that these transition probabilities T are simply estimates which are more or less accurate, depending on how much data is available. As an illustration, Tetreault and Litman used the following example [65]: in an MDP model, we have $S = \{S_1, S_2, S_3\}$, $A = \{A_1, A_2\}$. From a training corpus Γ, there were ten cases that an action A_1 was taken from state S_1. Out of these, three times the system transitioned back to state S_1, two times it transitioned to state S_2, and five times to state S_3. Thus we have

$$P(S_1|S_1, A_1) = \frac{3}{10} = 0.3 \tag{3}$$

$$P(S_2|S_1, A_1) = \frac{2}{10} = 0.2 \tag{4}$$

$$P(S_3|S_1, A_1) = \frac{5}{10} = 0.5 \tag{5}$$

From the same corpus, there were 1000 times that action A_2 was taken from state S_2. In 300 of those cases it transitioned to state S_1; in 200 cases to state S_2; and the remaining 500 times to state S_3. Thus,

$$P(S_1|S_2, A_2) = \frac{300}{1000} = 0.3 \tag{6}$$

$$P(S_2|S_2, A_2) = \frac{200}{1000} = 0.2 \tag{7}$$

$$P(S_3|S_2, A_2) = \frac{500}{1000} = 0.5 \tag{8}$$

While both sets of transition parameters have the same value, the second set is more reliable. In order to take reliability into account, Tetreault and Litman proposed a CI estimate based upon the available data [65]. It is done by taking a transition matrix T for a slice and sampling from each row using a Dirichlet distribution for q times ($q = 1000$ in this project). As a result, it generates a large number of new transition metrics $T_1, T_2, \cdots, T_q$ that are all very similar to T. They then run an MDP on all q transition matrices to get a range of ECR's. Both the ECR and the 95%CI are used for feature selection as described in section 4.

To summarize, for each given $< S, A, R >$ and selected training data Γ, an optimal pedagogical policy π can be induced by applying Tetreault, & Litman's toolkit. In this case, our RL approach is quite straightforward (see Algorithm 1).

Algorithm 1 Apply RL To Induce Policies with $< S, A, R >$ and Given Γ

```
Represent the training corpus $\Gamma$ with $< S, A, R >$.
Estimate the transition probabilities T.
Compute the optimal dialogue policy, $\pi$, based on $< S, A, R, T >$.
```

3.2. A Simple Example of Induced Policies

Figure 1 shows an example of an RL-induced policy. The policy is for Elicit/Tell decisions so we have $A = \{Elicit, Tell\}$ (shown in the second line in Figure 1) and we used the exploratory corpus as the training corpus $\Gamma_{Exploratory}$. The reward function R is students' $NLG \times 100$.

The first line in Figure 1 shows that the example policy involves only one state feature: [*StepSimplicityPS*], which is estimated from the training corpus based on the percentage of correct answers when the tutor has done an Elicit (i.e., asked a question) in a specific dialogue state. [1] Thus the higher the value of StepSimplicyPS, the easier the dialogue state. By only including one feature in the state, we assume that the decision to elicit or to tell should depend only on the simplicity level of the dialogue "state".

[S:] $= \{StepSimplicityPS\}$
[A:] $= \{Elicit, Tell\}$

[Policy:]
rule 1: [0] $\rightarrow$ Elicit
rule 2: [1] $\rightarrow$ Either Elicit or Tell

ECR: 8.09
95%CI: $[4.37, 12.07]$

Figure 1. Using StepSimplicityPS to Induce a Single Feature Policy on ET Decisions

The RL toolkit developed by Tetreault and Litman requires all state features in the model to be discrete variables. [*StepSimplicityPS*], however, is numeric and must be discretized before a suitable MDP can be constructed. In this project, the discretization procedure used

[1] Cordillera's dialogue manager is a finite state network, so dialogue "state" here refers to a state in that network. Many students will traverse the same network state, and some of those will leave the state via the tutor asking a question. This percentage refers to those students' answers.

two clustering procedures: the TwoStep procedure bounded the number of clusters in SPSS and the K-means procedure used K-means clustering to locate the optimal cluster centers. But other discretization procedures such as simple median split can also be applied.

After the discretization procedure, a continuous numeric variable [$StepSimplicityPS$] is binned into two binary values: 0 and 1. We have: "[$StepSimplicityPS$] : $[0, 0.38) \rightarrow 0; [0.38, 1] \rightarrow 1$", which means if StepSimplicityPS value is below 0.38, it is 0 (hard) otherwise, it is 1 (easy).

The number of MDP states in S determines the number of induced pedagogical rules in π. Because we only have one binary feature in the state representation, 2 pedagogical rules, rule 1 and rule 2, are induced (shown under the [*policy*] in Figure 1). Rule 1 says that when StepSimplicityPS is low (the content of this dialogue state is hard), the tutor should elicit; while rule 2 says when StepSimplicityPS is high, either elicit or tell will do. Figure 1 also shows that the example policy has: $ECR = 8.09$ (range $(-\infty, 100]$) with a 95% confidence interval $[4.37, 12.07]$, which means there is a 95% chance that the ECR of the learned policy is between a lower-bound of 4.37 and an upper-bound of 12.07.

So far, this section began with a description of how the problem of inducing pedagogical policies fits into the general RL and MDP framework. Then it described the induction toolkit employed and the assessment metrics used. In short, the promise of the MDP and RL approach is that once we build an estimated MDP that models the user population and learning context accurately, the induced policy should maximize the reward obtained from future students. With this approach, the problem of inducing effective pedagogical policies is thus reduced to computing the optimal policy for choosing actions in an MDP – that is, the tutoring system should take actions so as to maximize expected reward.

In this section, we described the abstract methodology by which pedagogical policies are induced when $< S, A, R >$ is defined and T is estimated from a given training corpus Γ. While this approach is theoretically appealing, the cost of obtaining human tutorial dialogues makes it crucial to limit the size of the MDP state space in order to minimize data sparsity problems, while retaining enough information in the states to represent accurately the human population and learning context. As mentioned above, our approach in this project is to begin with a large set of features to which a series of feature-selection methods were applied to reduce them to a tractable subset. More specifically, we applied 12 feature selection methods. In the next section, we will describe our feature selection procedures in details. In section 7, we will describe the instantiation of this methodology in Cordillera.

4. Applying Twelve Feature Selection Methods for Inducing Pedagogical Policies

To differentiate from the specific state representation S used in the RL and MDP formalism in section 2, we use Ω to represent a large set of potential state features. In other words, for any induced policy in this project, its corresponding state representation S is a subset of Ω. In the following, we will describe for a defined $< A, R >$ and a given system-student interactivity training corpus Γ how to select a subset S from a large feature choice set Ω that will generate the best policy.

In order to do feature selection, first we need to decide the maximum number of features, namely $\hat{m}$, to be included in the final state representation. In order to determine $\hat{m}$, it is necessary to consider the amount of available data and computational power. $\hat{m}$ should be small so that we have enough training data to cover each state, yet be large enough to include enough features to represent states without losing the information necessary to make good system decisions. In this project, for example, based on the minimum data available from the training corpora, we capped the number of features in each policy at six ($\hat{m} = 6$), which means that there can be as many as $2^6 = 64$ rules in the learned policy.

Before getting into the details of our twelve feature selection methods, we first define an initially empty set Θ for accumulating the induced policies for a defined $< \Omega, A, R >$ and a given training corpus Γ. The final best policy π^* will be selected from Θ by ECR. This is because ECR has been widely used as the criteria for evaluating induced policies, for example in the field of applying RL to induce policies from simulated corpora [34, 74, 73].

Almost every feature selection approach described below involves inducing single-feature policies first. That is, for each state feature choice in Ω, the RL toolkit was used to induce a single-feature policy such as the one used earlier as an illustration (see Figure 1). Although generating single-feature policies does not involve any feature selection procedure, it is counted as a feature selection method labeled as "single" for convenient reasons.

In the following, the focus is on using feature selection methods to induce policies with at least two features, referred to as multi-feature policies, and here we explored eleven feature-selection methods. Four were based upon the RL toolkit and were previously described in [14];

one was based on Principal Component Analysis (PCA); four were combinations of PCA and RL-based; and the final pair were based upon stochastic selection. Next, we will describe each approach in detail.

4.1. Four RL-based Feature Selection Methods

Detailed descriptions of the four RL-based approaches can be found in [14]. Here we will give a brief summary. As described above, Tetreault and Litman's toolkit calculates an optimal policy together with the policy's ECR and 95% CI [65]. Lower-Bounds and Upper-Bounds were used to refer to the 95% confidence bounds calculated for the ECR. For example, the example policy in Figure 1 has $ECR = 8.09$ with a 95% confidence interval= [4.37, 12.07].

To this point ECR has always been used as the criteria for selecting the best policies. However, a policy's Lower-Bound or Upper-Bound can also be used as the criteria. More specifically, the former evaluates the performance of policies in the worst case, while the latter describes how well the policy can perform. Additionally, we defined a new criterion named Hedge as: $Hedge = \frac{ECR}{UpperBound - LowerBound}$. Any of these criteria, ECR, Lower-Bound, Upper-Bound, or Hedge can be used to evaluate policies. Thus they are used as four different criteria for our RL-based feature selections. These feature-selection methods are fairly straightforward and use the same general procedure, described in Algorithm 2.

Algorithm 2 Four RL-based Feature Selection Procedure

```
for Ranking Metric in [ECR, Lower-Bound, Upper-Bound, Hedge] do
  Rank the features in $\Omega$ in descending order based upon the Ranking Metric of their corresponding single-feature-policies.
  for $i = 2$ to $\hat{m}$ do
    $S$ = the top i features from the ranked $\Omega$
    Induce a pedagogical policy $\pi$ with $< S, A, R >$ and $\Gamma$
    add $\pi$ to $\Theta$
  end for
end for
```

Based upon the ranking metrics used, the four RL-based feature-selection methods are named as ECR, Lower-Bound, Upper-Bound, and Hedge respectively. If we set $\hat{m} = 6$, then each of the four methods would add $\hat{m} - 1 = 5$ multi-feature policies to Θ.

4.2. PCA-based Feature Selection Method

Sometimes, the state features in Ω were highly correlated which reduced their expressiveness when used together. For example, three state features defined in this project are: the number of correct student responses namely "nCorrectKCPM", the number of incorrect student responses namely "nIncorrectKCPM", and the percentage of correct student responses namely "pctCorrectKCPM". The third variable can be easily calculated from the first two in that we have:

$$pctCorrectKCPM = \frac{nCorrectKCPM}{nCorrectKCPM + nIncorrectKCPM}$$

Therefore, it was necessary to apply an analysis procedure to avoid redundant features. Here we used Principal Component Analysis (PCA) [35]. More specifically, all state feature choices in Ω were first normalized; then PCA was applied to the normalized features to generate principal components and their corresponding eigenvalues. These eigenvalues were arranged in descending order, and all components whose eigenvalues were less than 1 were removed. For each eigenvalue, the feature that was maximally correlated with the corresponding principal component was identified.

The resulting features were a subset of Ω and thus we designated them the *PCA-feature subset*, labeled as Ω_{PCA}. Ω_{PCA} is an ordered list arranged by the eigenvalues of its corresponding principal components. Once Ω_{PCA} was identified, the PCA-only feature selection procedure was straightforward. It began with the first feature in Ω_{PCA} and added one feature at a time to induce a new policy. This process was repeated until the number of features in the state representation S reaches $\hat{m}$. Thus, for $\hat{m} = 6$, the PCA feature selection method would add ($\hat{m} - 1 = 5$) multi-feature policies to Θ.

4.3. Four PCA and RL-based Feature Selection Methods

Next four new feature selection approaches are created by simply combining PCA-only feature selection with the four RL-based feature selection methods. In these approaches, PCA is used to winnow Ω into Ω_{PCA} and then the four RL-based methods, *ECR*, *Upper_Bound*, *Lower_Bound*, and *Hedge* are applied on Ω_{PCA} only. The four feature selection methods were thus named PCA-ECR, PCA-LowerBound, PCA-UpperBound, and PCA-Hedge respectively. Similar to previous approaches, if we set $\hat{m} = 6$, then each of four PCA and RL-based combined feature selection methods adds $\hat{m} - 1 = 5$ multi-feature policies to Θ.

4.4. Random Feature Selection Methods

So far nine relatively straightforward feature selection methods have been introduced. In order to evaluate their relative effectiveness, two random feature selection methods were employed. The expectation was that the nine methods would be at least more effective than random selection.

The two random feature selection methods were named *Random* and *PCA-Random* respectively. This is because features were randomly selected from Ω and Ω_{PCA} respectively. Moreover, for a given $\hat{m}$, both random and PCA-random selections ran two rounds, generating $\hat{m}-1$ policies in each round. In other words, $2\times(\hat{m}-1)$ multi-feature policies were generated for either random method. If we set $\hat{m}=6$, then the two random selection methods would add in a total of 20 $(4\times(\hat{m}-1))$ multi-feature policies to Θ.

4.5. Summary on the General Feature Selection Procedure

To summarize, in this project we first defined a large set of features that represent relevant information about the learning environment and then applied twelve (including single) feature selection approaches to compress the learning environment to a small but carefully selected feature space. Our procedure can be summarized as:

1. Define $<\Omega, A, R>$ and choose Γ.
2. Decide $\hat{m}$ to be included in the final policy.
3. Set $\Theta = \emptyset$
    a) Inducing all single-feature policies and add them to Θ.
    b) Apply eleven feature selection methods to induce a total of $13\times(\hat{m}-1)$ multi-feature policies [2] and add them to Θ.
4. Select the best policy from Θ by ECR.

4.6. An Example of the Best Policy

In this project we defined fifty features in Ω (details in section 7). After running the twelve feature selection methods on Ω, a resulting “best” policy is shown in Figure 2. The feature selection method involved in inducing this policy is one of the four RL-based methods: *ECR feature selection.* Note that here we used the same $< A, R >$ and the same

[2] Either the random or PCA-random selection would result in $2\times(\hat{m}-1)$ multi-feature policies each and each of the rest nine feature selection would result in $(\hat{m}-1)$ policies.

training corpus $\Gamma_{Exploratory}$ as those used when inducing the single-feature policy in Figure 1. Thus, we have $A = \{Elicit, Tell\}$ (shown in the second line in Figure. 2), the reward function is students' $NLG \times 100$, and the training corpus is $\Gamma_{Exploratory}$.

Compared with the single-feature policy in Figure 1 which has only *StepSimplicityPS* in the state representation, the policy in Figure 2 has three state features. Their corresponding descriptions and discretization information are presented below:

[**StepSimplicityPS** $[0, 0.38) \rightarrow 0; [0.38, 1] \rightarrow 1$]**:** encodes a step's simplicity level. Its value is estimated from the training corpus based on the percentage of correct answers given on the dialogue state. The descretization procedure binarized the feature into two values: 0 and 1. If less than 38% of the answers given were correct, then this is considered to be 'hard' content and we set StepSimplicityPS =0; Otherwise, StepSimplicityPS = 1.

[**TuConceptsToWordsPS** $[0, 0.074) \rightarrow 0; [0.074, 1] \rightarrow 1$]**:** represents the ratio of the physics concepts to words in the tutor's utterances so far. The higher this value, the greater the percentage of physics content included in tutor turns. Dialogue states with less than 7.4% on this measure have TuConceptsToWordsPS=0 and 1 otherwise.

[**TuAvgWordsSesPS** $[0, 22.58) \rightarrow 0; [22.58, \infty) \rightarrow 1$]**:** encodes the average number of words in tutor turns in this session. This feature reflects how verbose the tutor is in the current session. The discretization procedure set the threshold at 22.58, so dialogue states when the tutor had above 22.58 words per tutor turn in the current session were represented with 1 for TuAvgWordsSesPS and 0 otherwise.

Since each of the three features was discretized into 2 values, a three-feature state representation would result in a state space of $2^3 = 8$. Thus, 8 pedagogical rules are learned. Figure 2 shows that in 5 situations the tutor should elicit (rules 1-5), in one situation it should tell (rule 6); in the remaining 2 cases either will do (rules 7-8).

For example, let's explain rule 6 since it is the only situation in which the tutor should tell. In rule 6, the state is $[0 : 1 : 0]$, which represents the values of the three corresponding features: StepSimplicityPS, TuConceptsToWordsPS and TuAvgWordsSesPS respectively. Rule 6 suggests that when the next dialogue content step is hard (as StepSimplicityPS is 0), the ratio of physics concepts to words in the tutor's entries is high (as TuConceptsToWordsPS is 1), and the tutor is not very wordy in the current session (as TuAvgWordsSesPS is 0), then the tutor should tell. As you can see, a three-feature policy is already quite subtle and adaptive to the learning context. Moreover, it is not like most of the tutorial policies derived from analyzing human tutorial dialogues.

[S:] $= \{StepSimplicityPS \times TuConceptsToWordsPS \times TuAvgWordsSesPS\}$
[A:] $= \{Elicit, Tell\}$

[Policy:]

rules 1-5: $\begin{bmatrix} 0:0:0 \\ 0:0:1 \\ 1:0:1 \\ 1:1:0 \\ 1:1:1 \end{bmatrix} \rightarrow$ **Elicit**

rule 6: $\begin{bmatrix} 0:1:0 \end{bmatrix} \rightarrow$ **Tell**

rules 7-8: $\begin{bmatrix} 0:1:1 \\ 1:0:0 \end{bmatrix} \rightarrow$ **Either Elicit or Tell**

ECR: 14.25
95%CI: $[10.04, 18.12]$

Figure 2. An Example Of Selected “best” Policy on ET Decisions

Figure 2 also shows that the induced three-feature policy has: $ECR = 14.25$ with a 95% confidence interval $[10.04, 18.12]$. It shows that adding *TuConceptsToWordsPS* and *TuAvgWordsSesPS* to the single state representation of *StepSimplicityPS* is effective. Compared with the single feature policy *StepSimplicityPS* in Figure 1, the three-feature policy in Figure 2 not only has higher ECR (14.25. vs. 8.09), but also has a higher lower-bound (10.04 vs. 4.37) and upper-bound (18.12 vs. 12.07). Thus, in theory it should be more effective than the single-feature policy.

5. General Approach

In the learning literature, it is commonly assumed that the relevant knowledge in domains such as math and science is structured as a set of independent but co-occurring Knowledge Components (KCs) and that KC’s are learned independently. A KC is “a generalization of everyday terms like concept, principle, fact, or skill, and cognitive science terms like schema, production rule, misconception, or facet" [68]. For the purposes of ITSs, these are the atomic units of knowledge. It is assumed that a tutorial dialogue about one KC (e.g., kinetic energy) will have no impact on the student's understanding of any other KC (e.g, of gravity). This is an idealization, but it has served ITS developers well for many decades, and is a fundamental assumption of many cognitive models [3, 49].

When dealing with a specific KC, the expectation is that the tutor's best policy for teaching that KC (e.g., when to Elicit vs. when to Tell) would be based upon the student's mastery of the KC in question, its intrinsic difficulty, and other relevant, but not necessarily known factors specific to that KC. In other words, an optimal policy for one KC might not be optimal for another. Therefore, one assumption made in this project is that ***inducing pedagogical policies specific to each KC would be more effective than inducing an overall KC-general policy***.

The domain chosen for this project is the Physics work-energy domain, a common component of introductory college physics courses. Two domain experts, who are also knowledge representation experts (not the authors), identified 32 KCs in the domain. They had experience identifying KCs for a series of previous studies involving college physics. Note that a complicated domain like physics can often be broken into many KCs. Here the 32 identified KCs are believed to cover the most important knowledge in the domain. In order to induce effective pedagogical policies, we investigated KC specific pedagogical policies in our RL applications.

This was a three-year project that can be divided into three stages, one stage per year since Fall 2007. In each stage, a group of students was trained on Cordillera. All three groups followed the same procedure: completing a background survey, reading a textbook, taking a pre-test, training on Cordillera, and finally, taking a post-test. All three groups used the training problems and instructional materials but on different versions of Cordillera. The versions differed only in terms of the pedagogical policies employed for interactive tutorial decisions.

In Stage 1, Cordillera made interactive decisions randomly and we collected an *exploratory corpus* that examined the consequences of each tutorial decision with real students. The student group is thus referred to as the Exploratory Group. In order to differentiate this version of Cordillera from the ones used in subsequent studies, this version is referred to as Random-Cordillera.

In Stage 2, we tried our first round of policy induction on all 32 KCs. Then these RL-induced policies were empirically evaluated. More specifically, Tetreault, & Litman's toolkit was applied to the Exploratory corpus to induce a set of pedagogical policies. Because we dichotomized the students' NLGs into +100 and −100 as reward functions, the induced policies were referred to as Dichotic Gain (DichGain) policies[15, 13]. The induced DichGain policies were added to Cordillera and this version of Cordillera was named DichGain-Cordillera. Except for following different policies (random vs. DichGain), the remaining components of Cordillera, including the GUI interface, the training problems, and the tutorial scripts, were left untouched. DichGain-Cordillera's effectiveness was tested by training a new group of 37 college students in 2008. Results showed that although the DichGain policies generated significantly different patterns of tutorial decisions than the random policy, no significant overall difference was found between the two groups on the pretest, posttest, or the NLGs [15].

There are many possible explanations for the lack of difference in learning outcomes between the DichGain and Exploratory groups. We argue that the exploration of our RL approach in stage 2 was limited. As described above, applying RL to induce effective tutorial policies may not be a simple task for which we can plug a training corpus into a toolkit. Rather it depends on many factors, such as state feature choices, feature selection methods, and the definition of reward functions. In stage 2, only 18 features were included in our state feature choices and no more than four appeared in the final induced tutorial policies. Thus the defined 18 features may be insufficient to adequately represent the state. Moreover our greedy-like feature selection process may also have limited our success. More details on the procedure of inducing DichGain policies and an example of DichGain policies can be found in [15]. Therefore stage 3 was designed to address these limitations in hopes of producing more effective pedagogical policies.

In stage 3, the approach to RL-related issues was greatly modified. Instead of focusing on all 32 KCs, we only focused on the eight primary KCs. We directly used students' Normalized Learning Gain (NLG)$\times$100 as the reward function instead of dichotomizing the NLG. The induced set of tutorial policies is thus named *Normalized Gain (NormGain)* tutorial policies and the version of Cordillera was named NormGain-Cordillera. We again ran a new group of students, named the NormGain group, using the same educational materials as used by the previous two groups. Next, we will describe the general methods involved in this project.

6. Method

In this section, we will first describe the NL tutoring system involved in this project, Cordillera. Then we will describe the two types of tutorial decisions, elicit/tell and justify/skip-justify, that the RL-induced pedagogical policies make. After that, we will describe the domain chosen for this project and specifically focus on the eight major KCs on which KC-specific NormGain policies were induced in Stage 3. Then we will describe the experimental procedure which is identical for all three studies in all three stages. Finally, we will describe our grading criteria for this project.

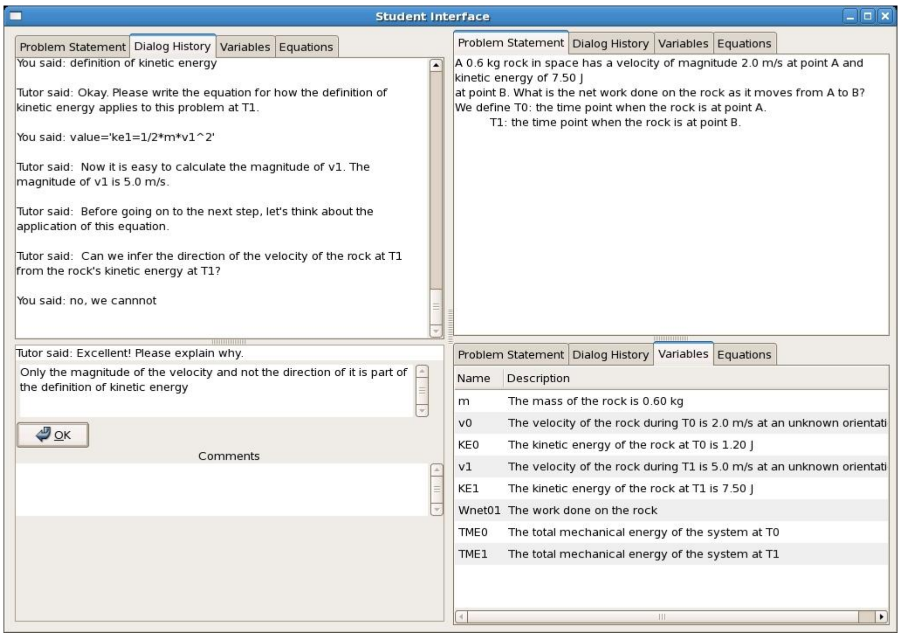
6.1. Cordillera

Cordillera is an NL Tutoring System that teaches students introductory college physics [68] and was developed using the TuTalk NL tutorial dialogue toolkit [37, 36]. TuTalk is an authoring tool which enables domain experts to construct natural language tutoring systems without programming. Instead, the domain experts focus on defining the tutoring content by writing tutorial scripts, which are then used for automating interactions. In other words, the script authors determine the flow of the dialogue and the content of each tutor turn.

The student interface is used by students to read the tutor's tutorial instructions and to answer questions by means of natural language entries. Figure 3 shows a screen shot of the student interface. The Message Window, located in the bottom-left corner is where the dialogue interaction takes place. The remaining panes are the Dialogue History Pane (upper-left), Problem Statement pane (upper-right), and Variable Pane (lower-right).

To reduce potential confounds due to imperfect NL understanding, the NL understanding module in Cordillera was replaced with a human interpreter called the language understanding wizard [9]. The *only* task performed by the human wizards is to match students' answers to the closest response from a list of potential responses and they cannot make any tutorial decisions. In this format, Cordillera works as a communications framework that connects a student interface to a wizard interface.

Across three stages of this project, three different versions of Cordillera were constructed, each of which differed only in terms of the pedagogical policies employed. The remaining components of the system, including the GUI interfaces and domain experts' tutorial scripts, were identical for all participants. In Cordillera the pedagogical policies are



You said: definition of kinetic energy

Tutor said: Okay. Please write the equation for how the definition of kinetic energy applies to this problem at T1.

You said: value='ke1=1/2*m*v1^2'

Tutor said: Now it is easy to calculate the magnitude of v1. The magnitude of v1 is 5.0 m/s.

Tutor said: Before going on to the next step, let's think about the application of this equation.

Tutor said: Can we infer the direction of the velocity of the rock at T1 from the rock's kinetic energy at T1?

You said: no, we cannnot

Tutor said: Excellent! Please explain why.

Only the magnitude of the velocity and not the direction of it is part of the definition of kinetic energy

A 0.6 kg rock in space has a velocity of magnitude 2.0 m/s at point A and kinetic energy of 7.50 J at point B. What is the net work done on the rock as it moves from A to B?
We define T0: the time point when the rock is at point A.
T1: the time point when the rock is at point B.

Name	Description
m	The mass of the rock is 0.60 kg
v0	The velocity of the rock during T0 is 2.0 m/s at an unknown orientati
KE0	The kinetic energy of the rock at T0 is 1.20 J
v1	The velocity of the rock during T1 is 5.0 m/s at an unknown orientati
KE1	The kinetic energy of the rock at T1 is 7.50 J
Wnet01	The work done on the rock
TME0	The total mechanical energy of the system at T0
TME1	The total mechanical energy of the system at T1

Figure 3. Cordillera Student Interface

used to make two types of tutorial decisions. Next, we will briefly describe them in detail.

6.2. Two types of Tutorial Decisions

Two types of tutorial decisions: Elicit/Tell (ET) and Justify/Skip-justify (JS) were the focus of our attention. For both ET and JS decisions, there is no widespread consensus on how or when either of these actions should be taken. This is why our research objective is applying RL to learn policies on them.

6.2.1. *Elicit/Tell*

During the course of one-on-one tutoring, the tutor often faces a simple decision, to *elicit* the next step from a student, or to *tell* a student the next step directly. We refer to such tutorial decisions as *elicit/tell (ET) decisions*. While a lecture can be viewed as a monologue consisting of an unbroken series of tells, human one-on-one tutoring is characterized by a mixture of elicits and tells. Some existing theories of learning suggest that when making tutorial decisions, a tutor should adapt its actions to the students' needs based upon their current knowledge level, affective state, and other salient features [53, 51, 22, 21, 70, 18, 40].

Typically, these theories are considerably more general than the specific interaction decisions that system designers must make. This makes it difficult to instantiate these theories as specific pedagogical policies in ITSs. Therefore when facing a decision whether to elicit or to tell a new step, most existing tutoring systems always decide to elicit [4, 41, 69, 27, 67, 44]. For example, existing NL tutoring systems often mimic a pervasive five-step dialogue pattern found in human tutoring that starts with the tutor posing a question or problem [26, 67].

Figure 4 presents a pair of sample dialogues comparing elicit and tell versions of a single tutorial dialogue extracted from the log files collected during this project. Both dialogues begin and end with the same tutor turns (lines 1 and 6 in (a) and 1 and 4 in (b)). However, in dialogue (a) the tutor chooses to elicit twice (lines 2-3 and 4-5 respectively) while in dialogue (b) the tutor decides to tell twice (lines 2 and 3). Note that the two dialogues cover the *same domain content.*

6.2.2. *Justify/Skip-justify*

The second tutorial decision was whether to execute a justification step, also referred to as self-explanation in much of the learning literature. During the tutoring process, human tutors sometimes ask students to *justify* a step they have taken or an entry they have made. Their apparent goal appears to be to help students understand domain knowledge in a deeper way. The open question is whether or not the tutor should conduct an elaborate discussion of a problem solving step given that this discussion is not necessary for the solution. We refer to such tutorial decisions as *justify/skip-justify (JS) decisions.*

Much previous research including [12], [19], and [2] found that asking students to justify their solution steps improves student learning. However, eliciting such a discussion may not always be desirable if, for example, the student is well aware of the rationale. If so, typing in a justification can be slow, frustrating, and distracting. Indeed, in domains like second language acquisition, Wylie et al. found that tutors asking students to justify did not lead to better learning outcomes but did significantly increase student training time when compared to a control group that was not asked to enter justifications [75]. Additionally, Katz, O'Donnell, and Kay [39] found that in some cases it may be better to delay the justifications until the problem has been solved, especially if the justification is abstract, plan-based, or lengthy.

Figure 5 presents a pair of sample dialogues comparing justify and skip-justify versions of a single tutorial dialogue extracted from project log files. In part (a), a justification is employed to guide the student (line 3-4); while in part (b), the justification is skipped.

(a) Elicit Version

1. **T:** So let's start with determining the value of v1.
2. **T:** Which principle will help you calculate the rock's instantaneous magnitude of velocity at T1? **{ELICIT}**
3. **S:** definition of kinetic energy
4. **T:** Please write the equation for how the definition of kinetic energy applies to this problem at T1 **{ELICIT}**
5. **S:** ke1 =0.5*m*v1ˆ2
6. **T:** From KE1 = 0.5*m*v1ˆ2, ···

(b) Tell Version

1. **T:** So let's start with determining the value of v1.
2. **T:** To calculate the rockś instantaneous magnitude of velocity at T1, we will apply the definition of kinetic energy again. **{TELL}**
3. **T:** Let me just write the equation for you: KE1 = 0.5*m*v1ˆ2. **{TELL}**
4. **T:** From KE1 = 0.5*m*v1ˆ2, ···

Figure 4. Elicit vs. Tell

6.3. The Domain and Eight Primary Knowledge Components

The domain chosen for this project, Physics work-energy problem solving, is a common component of introductory college physics courses. As mentioned before, two domain experts identified 32 KCs for this domain. However, in a domain like physics, solving a problem requires producing an argument, proof or derivation consisting of one or more inference steps; each step is the result of applying a domain principle, operator or rule. Thus the major domain principles are more challenging and important than other KCs since the student's overall learning performance depends more on learning domain principles. Therefore, when inducing NormGain policies, the decision was made to focus only

(a) Justify Version

1. **T:**Can we infer the direction of the velocity of the rock at T1 from the rock's kinetic energy at T1? **{ELICIT}**
2. **S:**nope.
3. **T:**Excellent! Please explain why. **{JUSTIFY, ELICIT}**
4. **S:**Only the magnitude of the velocity and not the direction of it is part of the definition of kinetic energy.
5. **T:**Excellent! Now that we know v1, $\cdots$.

(b) Skip-justify Version

1. **T:**Can we infer the direction of the velocity of the rock at T1 from the rock's kinetic energy at T1? **{ELICIT}**
2. **S:**nope.
3. **T:**Excellent! **{Skip-JUSTIFY}**
4. **T:**Now that we know v1, $\cdots$.

Figure 5. Justify vs. Skip-justify

on the eight primary KCs corresponding to the eight major domain principles.

The eight major principles in the domain are shown in Table I. In Table I, the first column lists its corresponding KC number. The second column describes the name of the principle. The last column is the formula or mathematical expression of the principle.

6.4. Procedure

All participants in this project experienced the same five standard phases: 1) background survey, 2) pre-training, 3) pre-test, 4) training, and 5) post-test. Unless specified explicitly in the following, the procedure, reading contents, training materials, GUI, and test items were identical across all groups and in each phase there were no time limits.

Table I. Major Principles of Work and Energy

KC	Principle description	Expressions
KC_1	Weight Law (w)	$W = mg$
KC_{14}	Definition of Work (W)	$W = Fdcos(\alpha)$
KC_{20}	Definition of Kinetic Energy (KE)	$KE = \frac{1}{2}mv^2$
KC_{21}	Gravitational Potential Energy (GPE)	$GPE = mgh$
KC_{22}	Spring Potential Energy (SPE)	$SPE = \frac{1}{2}kd^2$
KC_{24}	Total Mechanical Energy (TME)	$TME = KE + GPE + SPE$
KC_{27}	Conservation of Total Mechanical Energy (CTME)	$TME_1 = TME_2$
KC_{28}	Change of Total Mechanical Energy for Non-isolated Systems (TMENC)	$Net_W = TME_2 - TME_1$

The background survey asked students for demographic information such as gender, age, SAT scores, high school GPA, experience with algebra, calculus, physics, and other information.

Following the background survey, students read the physics textbook during the pre-training and took the pre-test. The physics textbook was only available during phase 2, pre-training.

In phase 4, students were first trained to solve a demonstration problem, which did not include physics content, on Cordillera. The sole purpose of this step was to familiarize them with the GUI interface. They then solved the same seven training problems in the same order on corresponding versions of Cordillera.

Finally, students took the post-test. The pre- and post-tests were identical in this project. Both contained a total of 33 problems selected from the Physics literature by two domain experts (not the authors). The 33 problems covered 168 KC applications. The tests were given online and consisted of both multiple-choice and open-ended questions. Open-ended questions required the students to derive an answer by writing or solving one or more equations. Once an answer was submitted, students automatically proceeded to the next question without receiving any feedback on the correctness of a response. Students were not allowed to return to prior questions.

As mentioned above, the reward function for the RL algorithm is the NLG. Therefore we used identical pre- and post-tests to avoid the need to factor out test differences. Students were not informed that the tests would be identical at any point; they received no feedback on their test answers or test scores; and the minimum time between the pre- and posttest was one week.

Overall, only three salient differences exist between the three groups:

1. The Exploratory group with a population of 64 was recruited in 2007; the DichGain group with a population of 37 was recruited in 2008; and the NormGain group with a population of 29 was recruited in 2009.
2. Random-Cordillera made random decisions and the DichGain-Cordillera and NormGain-Cordillera followed the induced DichGain and NormGain policies respectively.
3. A group of six human wizards were used by the Exploratory and DichGain groups; but only one of the six wizards was available for the NormGain group.

6.5. Grading

All tests were graded in a double-blind manner by a single experienced grader by mixing all students' test answers together. In a double-blind manner, the grader does not know the group to which each answer belongs nor the test, pre-test or post-test to which each answer belongs. For all identified relevant KCs in a test question, a KC-specific score for each KC application was given. We evaluated the student's competence in the following sections based on the sum of these KC-specific scores, named cumulative KC-specific scores. This is because the KC-specific pre- and post-test scores were used to define the reward functions when applying RL to induce KC-specific policies. Later analysis showed that the same findings held for other scoring rubrics. For comparison purposes all test scores were normalized to fall in the range of [0,1].

7. Inducing NormGain Pedagogical Policies

When inducing NormGain policies we mainly focused on the eight primary KCs, one for each major domain principle. Thus, the overall problem of inducing a policy for ET decisions and a policy for JS decisions is decomposed into 8 sub-problems of each kind, one per KC. Among the eight KCs, KC_1 does not arise in any JS decisions and thus only an ET policy was induced for it. For each of the remaining seven KCs, a pairs of policies, one ET policy and one JS policy, were induced. So we induced 15 KC-specific NormGain policies. During the tutoring process, there were some decision steps that did not involve any of the eight primary KCs. For them, two KC-general policies, an ET policy and a JS policy, were induced. To sum, a total of 17 NormGain policies were induced in stage 3.

Both inducing KC-general and KC-specific policies shared the same common procedure. First, we need to define $< \Omega, A, R >$, choose a training corpus Γ, and determine the maximum number of features, $\hat{m}$ included in the induced policy. Once these factors are decided, the general procedure for the 12 feature selection methods described in section 4 are followed. In this project, we used the same state feature choices in Ω, Γ, and $\hat{m}$ for inducing each of the 17 NormGain policies.

7.1. STATE REPRESENTATION SET Ω

As described above, an effective state representation should minimize state size while retaining sufficient relevant information about the learning context. In this project, Ω consists of only features that could be computed automatically or evaluated objectively, such as gender. Hand-annotated dialogue features were omitted as the tutor would require the features to be available in real time when the induced policies are employed. Next, we will describe the 50 feature choices in our Ω.

The 50 feature choices were defined based upon six categories of features considered by previous research [48, 8, 23] to be relevant for making tutorial decisions.

Autonomy (A) Features relate to the amount of work performed by the student in the dialogue. We defined five numeric features, which end with an 'A' in their names. For example, *[tellsSinceElicitA]* is one of the five features in this category. It refers to the number of tells the student has received since the last elicit prompt, irrespective of the KC involved. For example, tellsSinceElicitA = 2 means that two tell decisions have been made since the last elicit decision. This feature reflects how active a student may be currently, that is, how much work the student has performed recently.

Background (BG) features describe general background information about the student. The five Background Features include gender, age, Math SAT, Verbal SAT, and pre-test scores. None of these features change during training on Cordillera. All five background features end with "BG". One important note was that for DichGain group, the following features, gender, age, Math SAT, and Verbal SAT, were not available because of an administrative error.

Problem Solving Contextual (PS) features encode information about the current problem-solving context. All fifteen problem solving-related features end with 'PS.' For example; one feature defined in this category is *StepSimplicityPS*. It is used when inducing the single-feature policy in Figure 1 and the three-feature policy in Figure. 2). *StepSimplicityPS* is always in the range of $[0, 1]$. "StepSimplicityPS" $= 1$ means it is a easy step, whereas if it is close to 0, it means it is a difficult question.

Performance (PM) features describe information about student performance during the training. All twelve performance-related features end with "PM." For example, *pctCorrectKCPM* refers to the percentage of the student's correct entries on a KC. It is calculated by assessing all of the correct cases on the present KC in the student's entries divided by the total number of cases the present KC appeared in the student's entries. This feature reflects the student's overall competence on the current KC.

Student Dialogue (SD) features characterize students language. All Student Dialogue related features end with "SD." They are simple linguistic features that are computed from the student's dialogue turns. For example, one feature in this category is *stuAverageWordsSD*, which refers to the average number of words per student turn. This feature reflects how verbose the student was overall.

Temporal Situation (T) features encode time-related information about the problem-solving process. All three temporal situation features are numeric and end with a 'T' in their names. For example, one feature in this category is *durationKCBetweenDecisionT*. It refers to the time since the last tutorial decision was made on the current KC. This feature reflects how active a student's knowledge of the current KC is. If it is high, it means that the tutor has not mentioned the KC recently so the student's knowledge about the current KC may be less activated.

7.2. Three Training Corpora

The choice of training corpus Γ is a complex one. In stage 3, we have three training corpora available: the Exploratory corpus $\Gamma_{exploratory}$ consisted of 64 complete tutorial dialogues; the DichGain corpus $\Gamma_{DichGain}$ contained 37; and the combined corpus $\Gamma_{combined}$ comprised a total of 101 dialogues. $\Gamma_{exploratory}$ was collected for RL and designed to explore the feature space evenly and without bias. $\Gamma_{DichGain}$, by contrast, is similar to many other pre-existing corpora by following a set of specific pedagogical strategies. Inducing a successful policy from $\Gamma_{DichGain}$ would show the potential for applying RL to induce effective tutorial policies from most pre-existing data. $\Gamma_{combined}$, in theory, offers the benefits of both as well as an increased dataset.

Across three corpora, the total number of ET decisions in a tutorial dialogue ranged from 250 to 332 and we have: $M = 273.89, SD = 12.46$

in $\Gamma_{exploratory}$ and $M = 270.54, SD = 10.00$ in $\Gamma_{DichGain}$; the number of JS tutorial decisions ranged from 52 to 71, we have $M = 56.61, SD = 3.43$ in $\Gamma_{exploratory}$ and $M = 58.43, SD = 2.82$ in $\Gamma_{DichGain}$; the total number of the tutorial decisions regardless of decision types for each system-student interaction dialogue ranged from 288 to 372[3], we have $M = 305.48, SD = 14.01$ in $\Gamma_{exploratory}$ and $M = 307.57, SD = 12.45$ in $\Gamma_{DichGain}$.

On a KC by KC basis, however, the average number of tutorial decisions varies significantly across KCs: from as few as four on KC_1 to more than 80 on KC_{20}. The average number of tutorial decisions on elicit/tell (ET) and justify/skip-justify (JS) also varies across the eight primary KCs. There are only 4 ET decisions on KC_1 and more than 70 ET decisions on KC_{20}. Similarly, there are only 2 JS decisions for KC_{14} on average and more than 16 for KC_{21}. Overall, the ET tutorial decisions were much more frequent than the JS ones.

In this project, when inducing NormGain policies, rather than selecting one corpus a priori, all three were used. More specifically, a set of tutorial policies were derived from each training corpus separately and then the best policy from all sets were selected by ECR.

7.3. Maximum Number of Features $\hat{m}$

As mentioned above, in order to determine $\hat{m}$, it is necessary to consider the amount of available data and available computational power. In the worst case scenario, there were only 2 JS tutorial decision steps in the DichGain training corpus for KC_{14}. Therefore, based on the minimum data available from the three training corpora, we capped the number of features in each policy at six, $\hat{m} = 6$, which means that there are at least $2^6 = 64$ states in the learned policy. Alternatively, we could have used a flexible number for different KCs. However, it is not the case that learned tutorial policies with six features were most effective and instead the final 17 induced NormGain policies primarily have 3-5 features in their state representation. Only one of 17 final policies has six features. As shown earlier, the three-feature policy in Figure 2 is already quite subtle so it appears that six is a reasonable number.

Once $< \Omega, A, R >$, Γ, and $\hat{m}$ are determined we just need to follow the twelve feature selection procedures described in section 4 to induce the 17 NormGain policies. Inducing KC-specific policies can be seen as a special case of inducing KC-general policies. Therefore, we will start with our policy induction procedure on two KC-general policies first.

[3] overall decisions < ET decisions + JS decisions because on certain tutorial decision steps, the tutor makes both types of decisions: JS first and then ET. When we calculated the overall decisions, such a step was counted as one decision step.

7.4. Inducing Two KC-general Policies

When inducing KC-general pedagogical policies on ET or JS decisions, we have $A = \{Elicit, Tell\}$ or $A = \{Justify, Skip - Justify\}$ respectively. The reward function R is the same for inducing either action A, which is calculated based upon students' $NLGs$ in that we have: $R = NLG \times 100$. Moreover, the KC-specific features defined in Ω become KC-general features by taking into account all of the previous instances regardless of KC. For example, *nCorrectKCPM* becomes the number of correct responses on all the KCs instead of on a specific KC.

Once $< \Omega, A, R >$, Γ, and $\hat{m}$ are all defined, we can apply our twelve feature selection methods to get the best policy for corresponding action decision A. The overall procedure is described in Algorithm 3.

Algorithm 3 Induce Two KC-General Pedagogical Policy on ET and JS

```
for $Action$ in $[ET, JS]$ do
  if $Action = ET$ then
    $A = \{Elicit, Tell\}$
  else
    if $Action = JS$ then
      $A = \{Justify, Skip - Justify\}$
    end if
  end if
  $\Theta_A = \emptyset$
  for $\Gamma$ in $[\Gamma_{Exploratory}, \Gamma_{DichGain}, \Gamma_{Combined}]$ do
    Calculate reward function R $= NLG \times 100$
    $\Theta_{A,\Gamma}$ = Apply twelve feature selection on $< \Omega, A, R >$, $\Gamma$, and
    $\hat{m}$ (see Section 3)
    $\Theta_A = \Theta_A \cup \Theta_{A,\Gamma}$
  end for
  Select $\pi^*_A$ from $\Theta_A$ by ECR
  RETURN $\pi^*_A$
end for
```

As described in section 4, the general feature selection procedure would result in: one single-feature-policy for each feature choice in Ω and $13 \times (\hat{m} - 1)$ multi-feature policies for a given Γ. In this project, we defined fifty features in Ω and $\hat{m} = 6$. For ET decisions, a total of $50 + 13 * (6 - 1) = 115$ KC-general ET policies were induced from each training corpus. Taken together, all three corpora resulted in a total of $115 \times 3 = 445$ KC-general ET policies. The final best KC-general

ET policy was selected from the pool by ECR. For the purposes of this project, the highest ECR irrespective of the confidence bounds or hedging was selected. The same procedure was also followed for inducing the KC-general JS NormGain policy.

Inducing 15 KC-specifc NormGain policies followed the same general procedure as inducing the two KC-general NormGain policies except that $< \Omega, A, R >$ are KC-specific. Next, we will briefly describe the procedure.

7.5. Inducing 15 KC-specific Pedagogical Policies

In order to learn KC-specific policies, KC-specific $< \Omega, A, R >$ are needed. For example, to induce policies on the ET decisions involving KC_{20}, we consider the ET decisions involving KC_{20} only, the state representation for KC_{20} only, and use the learning gains on KC_{20} only. Therefore, we annotated our tutoring dialogues with the KCs covered by the content and action decisions with the KCs covered by each action.

A group of five individuals (including the first author) annotated each of the tutoring dialogues and action decisions with the relevant KCs. The KCs were drawn from the set of 32 identified KCs. For each of the seven training problems, there were at least two annotators. For each of 32 identified KCs, the final kappa was ≥ 0.77 which is fairly high given the complexity of the task. After each utterance in the system-student interaction dialogue was annotated with the corresponding KCs, a KC-specific Ω can be defined and a KC-specific A can be identified.

Additionally, a domain expert (not the authors) also mapped the pre-/post test problems to the sets of relevant KCs. So we can calculate students' KC-specific reward functions R defined as: $NLG_{KC_i} \times 100$.

Once KC-specific $< \Omega_{KC}, A_{KC}, R_{KC} >$ are defined, we applied a similar procedure as for inducing KC-general NormGain in Algorithm 3.

To summarize, to induce each of 17 NormGain policies, three training corpora, a space of fifty features, and twelve feature selection methods were explored and a total of $115 \times 3 = 445$ potential policies were generated. Each final NormGain policy was selected from its corresponding pool by ECR. For example, the three-feature example policy shown in Figure 2 is one of the final 17 NormGain policies. It is a KC-specific ET policy on KC_{20} that is induced from the Exploratory Corpus $\Gamma_{exploratory}$.

The resulting 17 NormGain policies were added to Cordillera yielding a new version of the system, named NormGain-Cordillera. In order to execute these tutorial policies, the dialogue manager needed to keep a record of the student's current states on each KC. Moreover, it also retained a KC-specific record for the tutorial decision steps. So when a tutorial decision step occurred, the dialogue manager first looked up the KC(s) involved in that step and then looked up the corresponding policies. When a tutorial decision did not involve any specific KCs, the dialogue manager followed the KC-general tutorial policies. Next, the induced tutorial policies were evaluated on real human subjects to see whether the students whose interactions were guided by the NormGain tutorial policies would out-perform those guided by random or DichGain ones.

8. Experimentally Evaluating the NormGain Policies

The goal of the evaluation reported below is twofold: first, to test whether our improved RL methodology and software produced more effective pedagogical strategies than either random policies or the DichGain policies; and second, to determine the characteristics of the NormGain policies.

8.1. Overall Learning Results

A one-way ANOVA showed that there were no significant differences among the three groups on overall training time: $F(2,122) = 1.831$, $p = .17$. More specifically, the average total training time (in minutes) across the seven training problems, was $M = 278.73$ min, $SD = 67.38$ for Exploratory group, $M = 294.33$ min, $SD = 87.51$ for DichGain group, and $M = 259.99$ min, $SD = 59.22$ for NormGain group.

After solving seven training problems on Cordillera, all three groups scored significantly higher in the posttest than pretest: $F(1,126) = 10.40$, $p = 0.002$ for the Exploratory group, $F(1,72) = 7.20$, $p = 0.009$ for the DichGain group, and $F(1,56) = 32.62$, $p = 0.000$ for the NormGain group respectively. The results suggested that the basic practices and problems, domain exposure, and interactivity of Cordillera might help students learn even from tutors with non-optimal pedagogical strategies.

A one-way ANOVA was used for comparing the learning performance differences among the three groups. While no significant pre-test score differences were found: $F(2,127) = 0.53, p = 0.59$, there were significant differences among the three groups on both post-test scores and NLG scores: $F(2,127) = 5.16, p = .007$ and $F(2,127) = 7.57, p = 0.001$ respectively. Figure 6 compares the three groups on the pre-test, post-test, and NLG scores. Moreover, a t-test comparison showed that the NormGain group out-performed the DichGain group on both post-test scores and NLG scores: $t(64) = 3.28, p = .002, d^4 = 0.82$ and $t(64) = 3.68, p = 0.000, d = 0.95$ respectively. Similar results were found between the NormGain and Exploratory groups: $t(91) = 2.76, p = .007, d = 0.63$ on post-test, and $t(91) = 3.61, p = 0.000, d = 0.84$ on NLG scores respectively.

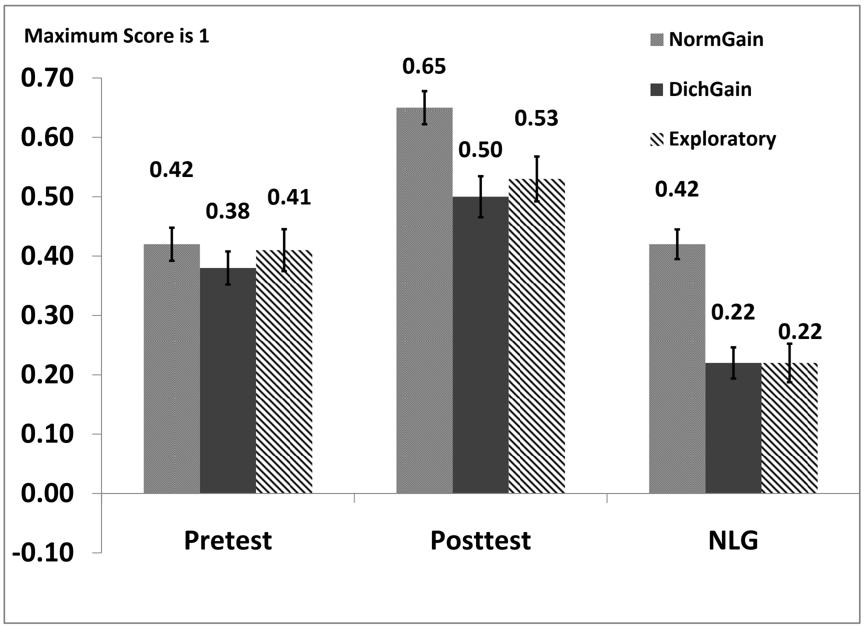


Figure 6. Compare Three Groups Learning Performance under Overall Grading

To summarize, the comparison among the three groups shows that the NormGain group significantly outperformed both the Exploratory and DichGain groups. These results were consistent both for the post-test scores and the NLGs and the effect sizes were large by Cohen's d criteria.

8.2. LOG ANALYSIS

Having compared the individual groups' learning performance, this subsection will compare the log file variations across the three groups.

[4] Cohen's d is defined as the mean learning gain of the experimental group minus the mean learning gain of the control group, divided by the groups' pooled standard deviation.

In order to quantify the amount of different decisions, we define an Interactivity ratio (I-ratio) and a Justification ratio (J-ratio).

I-ratio is defined as the number of elicit decisions a student received divided by the total number of ET decisions received in the entire tutorial dialogue. The higher this value, the more interactive the tutorial dialogue. Similarly, J-ratio is defined as the number of times the tutor executes a justification step divided by the total number of JS decisions the tutor made in a tutorial dialogue. The higher this value, the deeper and more elaborate the dialogue may be. Both values range from 0 (no elicits or justifies) to 1 (all elicitation or justification).

8.2.1. *I-Ratio*

Table II summarizes t-test comparisons on the I-ratio among the three tutorial corpora. In Table II, the first two columns list the two groups in comparison and their corresponding mean and SD scores. The last column lists the statistical results of the t-test comparisons. From Table II, the I-ratios for the three student groups were: 0.76 (NormGain), 0.44 (DichGain), and 0.50 (Exploratory) respectively and the differences among them were significant: we have $NormGain > Exploratory > DichGain$. Therefore, the NormGain policies seemingly resulted in more interactive tutorial dialogue than either the random or the DichGain policies.

Table II. Pairwise Comparison Among Three Groups On I-ratio

Group 1		*Group 2*		*Group 1 vs. Group 2*
NormGain	0.76 (0.07)	Exploratory	0.50 (0.03)	$t(91) = 24.72, p = 0.000$
NormGain	0.76 (0.07)	DichGain	0.44 (0.04)	$t(64) = 22.08, p = 0.000$
Exploratory	0.50 (0.03)	DichGain	0.44 (0.04)	$t(99) = 7.967, p = .000$

8.2.2. *Justify Ratio*

Similarly, Table III summarizes t-test comparisons on J-ratio among the three tutorial corpora. In Table III, the first two columns list the two groups in comparison and their corresponding mean and SD scores. The last column lists the statistical results of the t-test comparisons. Table III, shows that the mean of J-ratios for the three student groups were: 0.82 (NormGain), 0.43 (DichGain), and 0.53 (Exploratory) and the pair wise t-test comparisons show that on J-ratio, we have: $NormGain > Exploratory > DichGain$.

To summarize, the NormGain policies produced substantially more elicit and more justifications than the random or DichGain ones. In the next section the discussion will focus on some general characteristics of the induced tutorial policies. The 17 induced NormGain tutorial policies will be described by identifying the training corpus that each final tutorial tactic was derived from, which feature categories were most frequently involved in the final tutorial policies, and which feature selection method discovered the most final tutorial policies.

8.3. The Impact of Induction Decisions on the 17 NormGain Policies

The purpose of this section is to determine how the RL-related induction decisions described in the previous sections impacted the induced tutorial policies. For example, one decision was made to use all three training corpora; did the final induced NormGain policies come from one corpus or from all three corpora? Moreover, which features appeared in the final induced NormGain policies? Which feature selection method(s) seemed to be more effective? This section begins with a discussion of the training corpus involved in the final 17 NormGain tutorial policies.

8.3.1. *Source Training Corpus,* Γ

Table IV shows the source training corpus used to induce each of the 17 NormGain tutorial policies. As described above, among the 17 NormGain policies, two were KC-general policies, an ET policy and a JS policy. All of the remaining 15 NormGain policies are KC-specific policies; eight were on ET decisions, one for each of eight major domain principles, and seven were on JS decisions, one for each major domain principle except for KC_1. KC_1 does not arise in any JS decisions.

In Table IV, the first column lists the row number. The second column lists the KC on which the induced NormGain policy is specific. The third and fourth columns show the source training corpus used in

Table III. Pairwise Comparison Among Three Groups On J-ratio

Group 1		*Group 2*		*Group 1 vs. Group 2*
NormGain	0.82 (0.07)	Exploratory	0.53 (0.06)	$t(91) = 18.95, p = 0.000$
NormGain	0.82 (0.07)	DichGain	0.43 (0.07)	$t(64) = 22.85, p = .000$
Exploratory	0.53 (0.06)	DichGain	0.43 (0.07)	$t(99) = 7.894, p = .000$

Table IV. The Source Training Corpus Of the Induced 17 NormGain Tutorial Policies

		ET	JS
1	KC_1	$\Gamma_{DichGain}$	–
2	KC_{14}	$\Gamma_{Combine}$	$\Gamma_{Exploratory}$
3	KC_{20}	$\Gamma_{Exploratory}$	$\Gamma_{Exploratory}$
4	KC_{21}	$\Gamma_{Exploratory}$	$\Gamma_{Exploratory}$
5	KC_{22}	$\Gamma_{Exploratory}$	$\Gamma_{Exploratory}$
6	KC_{24}	$\Gamma_{DichGain}$	$\Gamma_{Exploratory}$
7	KC_{27}	$\Gamma_{Exploratory}$	$\Gamma_{Exploratory}$
8	KC_{28}	$\Gamma_{DichGain}$	$\Gamma_{DichGain}$
9	KC-general	$\Gamma_{Exploratory}$	$\Gamma_{DichGain}$

deriving NormGain tutorial policies on ET and JS for corresponding KCs respectively. For example, the cell "$\Gamma_{DichGain}$" in the first row shows that the final NormGain ET policy on KC_1 is induced by using $\Gamma_{DichGain}$. While the last cell in the first row is empty because KC_1 does not arise in any JS decisions and thus no corresponding policy was induced.

Table IV shows that all three Corpora were involved in generating the final tutorial policies. However, the majority of the NormGain tutorial policies were induced by using $\Gamma_{Exploratory}$, 11 (5 ET and 6 JS) out of 17; $\Gamma_{DichGain}$ was used to generate five (3 ET and 2 JS); and $\Gamma_{Combine}$ only generated one NormGain policy. It is not very intuitive to determine why most of the NormGain policies were from $\Gamma_{Exploratory}$. Future work is needed to explore the characteristics of a training corpus and how to choose a training corpus. However, this result reinforces the importance of collecting an Exploratory Corpus when applying RL to induce effective pedagogical policies.

8.3.2. *Feature Selection*

To induce each of the 17 NormGain policies, we applied 12 feature selection methods on fifty feature choices. It would be interesting to see which feature selection method(s) found the most final NormGain tutorial policies. Table V lists all the feature selection methods that were followed to get the final 17 NormGain tutorial policies. "single" means it is a single feature policy.

In Table V, the first column lists the row number. For example, the ninth row shows that the final KC-general NormGain policy on ET decisions is induced through applying "Lower-Bound" feature selection while the KC-general NormGain policy on JS decisions is induced through applying "ECR" feature selection.

From Table V, it can be concluded that the five feature selection approaches: PCA-only, PCA-ECR, PCA-UpperBound, PCA-LowerBound, and random did not elicit any of the final tutorial policies. All other six approaches resulted in at least one. Among them, the two RL-based feature selection methods appeared to be most effective. The Upper_Bound method found five NormGain tutorial policies and the ECR based method discovered four NormGain tutorial policies.

Previously, we explored four RL-based feature selection methods on 18 features in [14]. That work also showed that the Upper-Bound selection seemed to be the best among the four. When inducing NormGain policies, we explored the four RL-based feature selection methods together with eight other methods on 50 features. Our results here seems to be consistent with our previous results in [14] in that the Upper_Bound method seems to be an effective feature selection method in applying RL to induce pedagogical policies.

One possible explanation may be due to our pedagogical goal. In this project, we were mainly interested in improving student learning. Generally speaking, learning occurs with low frequency but also with low risk. In other words, while learning is difficult to achieve, the risk of not learning is also low (no learning gains). However when learning does occur, the reward is always positive. Therefore, it is reasonable for Upper_Bound to be a more effective feature selection method since it can take risks toward selecting features that can cause learning to occur. In short, this result indicates that in education, features that may result in higher learning gains should always be considered by the tutor when making decisions. This is likely due to the fact that in the worst case a student will simply not learn rather than lose information so the cost of considering superfluous features is low.

Overall, the results show the relative effectiveness of our twelve feature selection methods in that they at least beat the random feature selection. However, our feature selection may still need to be improved because one of the final induced policies is from the PCA-random feature selection method — the ET NormGain policy on KC_{27}.

8.3.3. *Feature Choices*

Table VI summarizes features appearing in each of 17 NormGain policies. The first column lists the row number. The second and third columns show the corresponding KC and action decisions. The fourth column lists the number of features involved in the corresponding Nor-

Table V. Applying 11 Feature Selection Methods to Induce 34 Tutorial Policies

		ET	JS
1	KC_1	single	–
2	KC_{14}	single	single
3	KC_{20}	ECR	PCA-Hedge
4	KC_{21}	Upper_Bound	PCA-Hedge
5	KC_{22}	Hedge	Upper_Bound
6	KC_{24}	ECR	Upper_Bound
7	KC_{27}	PCA-Random	ECR
8	KC_{28}	Upper_Bound	Upper_Bound
9	KC-general	Lower_Bound	ECR

mGain policies and the last column lists all the feature names in the corresponding policy's state representation.

For illustration purpose, we used simplified variable names here that mainly indicate the category of the features. Recall that the six categories are: Autonomy (A) features, background (BG) related feature, Performance (PM) related features, Problem Solving (PS) Contextual features, Student Dialogue (SD) features, and Temporal Situation (T) features.

For example, row 10 is about the JS NormGain policy on KC_{24}. Row 10 shows that there are six features, "v14PS v18PS v49BG v59SD v60SD v62SD", in the state representation for this policy. Among the six features, "v14PS" and "v18PS" are Problem Solving Contextual features since their name ends with "PS"; "v49BG" is a background feature; and the remaining three features are all Student Dialogue (SD) features.

From Table VI, we can see that certain state features were involved in multiple NormGain policies. For example, "v18PS" represents the state feature *StepSimplicityPS*. Table VI shows that *StepSimplicityPS* appears in 7 NormGain policies as "v18PS" is listed in row 4, 6, 7, 10, 11, 15, and 16. On the other hand, not every feature choice in Ω occurred in the final 17 induced NormGain policies. For example, "v44PM" represents the feature *pctCorrectKCPM* (percentage of correct student entries involving the current KC), which was not involved in any NormGain Policy. In fact, only 30 out of 50 features occur in at least one induced NormGain policy.

Table VI. Features Involved in 17 NormGain Tutorial Policies

	KC	action	#features	Features
1	KC_1	ET	1	v24PS
2	KC_{14}	ET	1	v5T
3	KC_{14}	JS	1	v12PS
4	KC_{20}	ET	3	v18PS v23PS v26PS
5	KC_{20}	JS	5	v9T v15PS v28A v56SD v62SD
6	KC_{21}	JS	3	v13PS v18PS v20PS
7	KC_{21}	ET	3	v15PS v18PS v30A
8	KC_{22}	JS	5	v23PS v24PS v27A v46PM v47PM
9	KC_{22}	ET	2	v23PS v27A
10	KC_{24}	JS	6	v14PS v18PS v49BG v59SD v60SD v62SD
11	KC_{24}	ET	4	v10T v14PS v18PS v55SD
12	KC_{27}	ET	4	v5T v17PS v23PS v35PM
13	KC_{27}	JS	4	v9T v25PS v27A v56SD
14	KC_{28}	JS	3	v15PS v24PS v54SD
15	KC_{28}	ET	5	v5T v16PS v18PS v29A v41PM
16	$KC - general$	ET	4	v5T v18PS v27A v46PM
17	$KC - general$	JS	5	v16PS v17PS v23PS v25PS v27A

By summing column 4 in Table VI, we see that the total number of feature occurrences across 17 tutorial policies was 59. For illustration reasons, Table VII lists the number of features defined in each of the six categories and the feature occurrences in the final 17 NormGain policies. For example, the third and fourth columns in row 4 in Table VII show that there are fifteen PS features defined and they account for thirty out of 59 feature occurrences in the final 17 tutorial policies. In other words, more than half of all feature occurrences in the 17 NormGain policies were from PS.

Table VII shows that both the number of features defined for each category and the feature occurrences in the six categories is not even. However, we argue that the uneven distribution of the feature occurrences among the six categories was not caused by the uneven number of features defined in each category.

Across the fifty features, the most frequent feature appears seven times. Four features appear in more than three induced policies and they are:

Table VII. Occurrence of Six Category Features in The Final NormGain Tutorial Policies

		Defined Features	Feature Occurrences
1	Autonomy(A)	5	8
2	Background(BG)	5	1
3	Performance(PM)	12	5
4	Problem Solving Contextual(PS)	15	30
5	Student Dialogue(SD)	10	8
6	Temporal Situation(T)	3	7
7	Total	50	59

StepSimplicityPS (7 Occurrences): labeled as "v18PS" in VI. It is a Problem Solving Contextual feature which encodes a step's simplicity level and its value is roughly estimated from the Combined Corpus based on the percentage of answers that were correct for a step.

TuConceptsToWordsPS (5 Occurrences): labeled as "v23PS" in VI. It is a Problem Solving Contextual feature which represents the ratio of physics concepts to words in the tutor's dialogue.

tellsSinceElicitA (5 Occurrences): labeled as "v27A" in VI. It is an Autonomy feature which represents the number of tells the student has received since the last elicit.

durationKCBetweenDecisionT (4 Occurrences): labeled as "v5T" in VI. It is a Temporal Situation feature which represents the time since the last tutorial decision was made on the current KC.

While *StepSimplicityPS* can be seen as a domain-oriented feature, the remaining three features are primarily system-behavior related features. The high occurrence of *StepSimplicityPS* in the NormGain policies is not very surprising because it is widely believed that difficulty level is an important factor in a system behaving adaptively and effectively. The frequent involvement of system-behavior related features in the induced policy may be because these features could reflect a student's general aptitude and the degree to which their knowledge on a specific KC is activated. For example, *tellsSinceElicitA* reflects how interactive a student has been recently and durationKCBetweenDecisionT reflects how active a student's knowledge on the current KC is. When *durationKCBetweenDecisionT* is high, it means that the tutor has not mentioned the KC recently so the student's knowledge on the current KC may be less activated.

Much to our surprise, the features related to the students' overall or recent performance and background (e.g., MSAT, VSAT, gender, pretest score) appeared the least in the NormGain policies. Row 4 in Table VII shows that the twelve PM feature choices account for only five occurrences of this category in the final 17 NormGain policies. They seem to be less involved even than the Temporal Situation (T) related features. With only three features in Temporal Situation category, they account for 7 feature occurrences.

Among the twelve PM features, *nIncorrectKCPM* (the number of incorrect responses in the student's dialogue so far) is the most frequently occurring feature in that it appeared in two final NormGain tutorial policies. A feature such as *pctCorrectKCPM* (percentage of correct student's entries involving the current KC) did not appear in any of the final tutorial policies. Additionally, only one out of five background features occurred in one final tutorial tactic: *ageBG*** (the age of the student). The remaining four background features, such as MSAT, VSAT, gender, pretest score, were not involved in the final 17 NormGain policies.

To summarize, Problem Solving Contextual Features occurred most frequently, thirty times, in the final 17 induced tutorial policies. The features related to the students' overall or recent performance and background appeared the least in the NormGain policies. Although space does not permit a detailed discussion of the prevalence of features, it appears to be a mixture of easily anticipated dependencies (e.g., step simplicity) and a few surprises (e.g., students' overall and immediate performances directly reflect their level of knowledge. So why don't these factors matter?).

9. Discussion

To summarize, we described a practical methodology for using RL to improve the effectiveness of an ITS over time. In a nutshell, our RL methodology is to:

1. Choose an appropriate reward measure for the instructional goals, an appropriate list of features for the state representations, and identify a set of reasonable system decisions for which a policy is needed.

2. Build an initial training system that collects an exploratory dataset (one that tries many times from each state each of the actions between which the policy will decide). Despite being exploratory, this system should still provide the desired basic functionality.

3. Apply feature selection methods when necessary, based on the size of the exploratory corpus, to select a subset of features that capture the most effective factors in the learning environment. Then we use the exploratory corpus to build an empirical MDP model for the subset of state features. The transitions of this MDP model the user population's reactions and rewards for the various system action sequences.

4. Compute the optimal dialogue policy according to this learned MDP.

5. Add the learned policy to the system and evaluate the policy on a new group of users.

In this project, we investigated pedagogical skills at a micro-step level. i.e. pedagogical tutorial tactics. These tactics do not govern the domain solution path selected for presentation or the problems presented. They only govern low-level tutorial interactions, e.g. whether the student is told what principle to apply or whether the system elicits it with a prompt, and whether a student, once he/she has made a step, is asked to justify his/her answer. If fine-grained pedagogical skills of this type turn out to be effective, then more complex or content-oriented tactics, such as problem or sub-problem selection may be similarly effective.

Compared with previous studies on applying RL to induce pedagogical policies for ITSs, this project makes at least three major contributions. First, we bypassed the need for building simulated students by collecting an exploratory corpus; moreover, we showed that a relatively small exploratory corpus is enough for inducing effective policies. Second, we empirically showed that the RL induced policies indeed help students learn better. Third, while much of the previous research on applying RL to ITSs used pre-defined state representations, we defined a large set of features Ω to which several feature-selection methods were applied to reduce them to a tractable subset.

Moreover, we believe that this project also contributes to applying RL to induce dialogue policies for dialogue systems. We showed that using a relatively small exploratory corpus, model-based RL methods such as Policy Iteration can still induce effective policies even when the reward function is much delayed and the task domain is rather complex. We argue that this is done by a rather comprehensive definition of the state feature choices and exploration of various feature selection methods.

More specifically, we showed that RL is able to effectively search a very large continuous space of dialogue policies (after being discretized, the space is $\geq 2^{50}$ in size) using a relatively small amount of training dialogue data (64 subjects in the Exploratory group and 37 in the DichGain group). A post-hoc comparison showed that our learned policy outperformed both sets of training policies in terms of learning performance. Our results demonstrate that the application of RL allows one to optimize pedagogical strategies by searching through a much larger search space than can be explored with more traditional methods. This success supports the hypothesis that RL-induced rules are effective and that the approach taken in this project was a feasible one.

However, inducing effective tutorial policies was not trivial. The DichGain tutorial policies did not seem to be more effective than the random decisions in Random-Cordillera. A number of factors were changed in deriving NormGain policies compared to inducing DichGain policies. These changes included the feature choices, the choice of training corpora, feature selection methods and the definition of reward functions. So it is still not clear which factor or factors caused the change in effectiveness.

We also note that our learned polices made dialogue decisions based on Problem Solving Contextual features in conjunction with other features such as Autonomy features, and also made very subtle decisions compared with existing learning theories. As such, our RL-induced policies are not the standard pedagogical strategies investigated in the learning literature. For instance, it is widely believed that effective tutors adapt their behavior to the individual student knowledge level and incoming competence [70, 17, 40]. Indeed, individualized tutoring is considered a Grand Challenge by the National Academy of Engineering. However, such features appeared to play little role in the effective tutorial policies induced from our data. Rather it appears that the Problem Solving Contextual features are most involved in the final induced NormGain tutorial policies. Overall it appears that the learning context features that make the most difference for determining when to tell vs. elicit and when to Justify vs. Skip-Justify are not always the ones that first come to mind given current theories of learning and tutoring. Overall, our results suggest that when building an accurate learning context model, adding domain-oriented and system behavior related features would be beneficial.

Finally, our approach has begun to address some of the challenges of the prevailing theory and application of RL (e.g., balancing the competing concerns of random exploration with user experience in the training corpus; keeping the state space as small as possible in order to make learning data-efficient while retaining enough information for decision-making; and providing a general methodology for reducing the state space to a manageable size). Our RL approach showed that most of the NormGain tutorial policies were derived from the Exploratory Corpus and among the twelve feature selection methods tried in this project, the two RL-based feature selection methods, Upper-Bound and ECR, were most effective. However, in order to investigate why these are the case, we need more exploration. Additionally, one of the issues we would like to investigate is how different choices of training corpora or feature selection methods are correlated with learning gains. Also note that we keep the training material and experimental procedure identical for all three studies across all three stages, therefore it is not clear whether the induced policies would still be valid if some aspect of the training material or experimental procedure changes.

In this project, the induced NormGain policies is at best an approximation, we may be introducing the problem of hidden state or partial observability into the problem of choosing optimal tutorial actions in each state. For situations with hidden state a richer Partially observable Markov decision process (POMDP) model is often more appropriate [28]. In future work, we wish to explore POMDP as POMDPs allow for realistic modeling of the student's knowledge levels, the student's intentions, and other hidden state components by incorporating them into the state space.

Acknowledgements

NSF (#0325054) supported this work. We also thank the Learning Research and Development Center at the University of Pittsburgh for providing all the facilities used in this work.

10. Author Biographies

1. Dr. Min Chi

   Machine Learning Department, Carnegie Mellon University, 5000 Forbes Ave. Pittsburgh PA, 15213, USA

   Dr. Chi is a Post-Doctoral Fellow in the Machine Learning Department and Pittsburgh Science of Learning Center at Carnegie Mellon University. Dr. Chi received her B.A. degree in Information Science and Technology from Xi'an Jiaotong University and in 2009 received her Ph.D. degree in Intelligent Systems from the University of Pittsburgh. Her research focuses on the applications of machine learning techniques to interactive learning environments. In particular, she has focused on applying machine learning techniques to induce pedagogical strategies that can automatically adapt effectively to differences in students's performances, the subject matter, and instructional goals. She is also interested in comparing machine-learned pedagogical strategies to some of the existing learning theories.

2. Dr. Kurt VanLehn

   School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, USA

   Dr. VanLehn is a Professor of Computer Science at Arizona State University. He completed his Ph.D. at MIT, did post-doctoral research at Xerox PARC, joined the faculty of Carnegie-Mellon University in 1985, moved to the University of Pittsburgh in 1990 and joined ASU in 2008. His work involves applications of artificial intelligence to education, including intelligent tutoring systems, cognitive modeling and educational data mining.

3. Dr. Diane Litman

   Department of Computer Science, 5105 Sennott Square, 210 South Bouquet Street, University of Pittsburgh Pittsburgh PA, 15260, USA

   Learning Research and Development Center, Room 741, University of Pittsburgh, 3939 O'Hara St, Pittsburgh PA, 15260, USA

   Dr. Diane Litman is presently Professor of Computer Science, Senior Scientist with the Learning Research and Development Center, and faculty in Intelligent Systems, all at the University of Pittsburgh. Previously she was a member of the Artificial Intelligence Principles Research Department, AT&T Labs - Research (formerly Bell Laboratories), and an Assistant Professor of Computer Science at Columbia University. Dr. Litman received her B.A. degree in Mathematics and Computer Science from the College of William and Mary, and her M.S. and Ph.D. degrees in Computer Science from the University of Rochester. Dr. Litman's current research focuses on enhancing the effectiveness of intelligent tutoring systems through spoken language processing, affective computing, and machine learning.

4. Dr. Pamela W. Jordan

   Learning Research and Development Center, University of Pittsburgh, 3939 O'Hara St, Pittsburgh PA, 15260, USA

Dr. Jordan is a Research Associate at the University of Pittsburgh, working in the area of natural language tutorial dialogue. She received her B.S. in Computer Science from the University of Virginia, M.S. in Computer Science from George Mason University, M.S. in Computational Linguistics from Carnegie Mellon University and Ph.D. in Intelligent Systems from University of Pittsburgh. She completed her Ph.D. under the supervision of Richmond Thomason, in the area of natural language discourse and dialogue. The joint research described in this volume reflects her interest in building tutorial dialogue systems and understanding how educational dialogues impact human learning.

References

1. Hua Ai and Diane J. Litman. Knowledge consistent user simulations for dialog systems. In *Proceedings of Interspeech-2007*, pages 2697–2700, Antwerp, Belgium, 2007.
2. Vincent Aleven, Amy Ogan, Octav Popescu, Cristen Torrey, and Kenneth R. Koedinger. Evaluating the effectiveness of a tutorial dialogue system for self-explanation. In Lester et al. [42], pages 443–454.
3. John R. Anderson. *The architecture of cognition.* Cambridge, Mass. : Harvard University Press, 1983.
4. John R. Anderson, Albert T. Corbett, Kenneth R. Koedinger, and Ray Pelletier. Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, 4(2):167–207, 1995.
5. Ryan Shaun Baker, Albert T. Corbett, and Kenneth R. Koedinger. Detecting student misuse of intelligent tutoring systems. In Lester et al. [42], pages 531–540.
6. Ryan Shaun Baker, Albert T. Corbett, Kenneth R. Koedinger, and Angela Z. Wagner. Off-task behavior in the cognitive tutor classroom: when students "game the system". In Elizabeth Dykstra-Erickson and Manfred Tscheligi, editors, *CHI*, pages 383–390. ACM, 2004.
7. Tiffany Barnes and John C. Stamper. Toward automatic hint generation for logic proof tutoring using historical student data. In Beverly Park Woolf, Esma Aïmeur, Roger Nkambou, and Susanne P. Lajoie, editors, *Intelligent Tutoring Systems*, volume 5091 of *Lecture Notes in Computer Science*, pages 373–382. Springer, 2008.
8. Joseph Beck, Beverly Park Woolf, and Carole R. Beal. Advisor: A machine learning architecture for intelligent tutor construction. In *AAAI/IAAI*, pages 552–557. AAAI Press / The MIT Press, 2000.
9. Nielsole Ole Bernsen and Laila Dybkjaer. *Designing Interactive Speech Systems: From First Ideas to User Testing.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.
10. Nicoletta Calzolari, Claire Cardie, and Pierre Isabelle, editors. *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings*

of the Conference, Sydney, Australia, 17-21 July 2006. The Association for Computational Linguistics.

11. Iadine Chadés, Marie-José Cros, Frédérick Garcia, and Régis Sabbadin. Markov decision process (MDP) toolbox v2.0 for MATLAB. http://www.inra.fr/internet/Departements/MIA/T/MDPtoolbox, 2005.
12. Michelene T. H. Chi, Nicholas de Leeuw, Mei-Hung Chiu, and Christian LaVancher. Eliciting self-explanations improves understanding. *Cognitive Science*, 18(3):439–477, 1994.
13. Min Chi. *Do Micro-Level Tutorial Decisions Matter: Applying Reinforcement Learning To Induce Pedagogical Tutorial Tactics.* PhD thesis, Intelligent Systems Program, University of Pittsburgh, Dec. 2009.
14. Min Chi, Pamela W. Jordan, Kurt VanLehn, and Moses Hall. Reinforcement learning-based feature selection for developing pedagogically effective tutorial dialogue tactics. In Ryan Shaun Joazeiro de Baker, Tiffany Barnes, and Joseph E. Beck, editors, *The 1st International Conference on Educational Data Mining (EDM)*, pages 258–265, Montreal, Québec, Canada, 2008. www.educationaldatamining.org.
15. Min Chi, Pamela W. Jordan, Kurt VanLehn, and Diane J. Litman. To elicit or to tell: Does it matter? In Vania Dimitrova, Riichiro Mizoguchi, Benedict du Boulay, and Arthur C. Graesser, editors, *AIED*, pages 197–204. IOS Press, 2009.
16. Min Chi, Kurt VanLehn, Diane J. Litman, and Pamela W. Jordan. Inducing effective pedagogical strategies using learning context features. In Paul De Bra, Alfred Kobsa, and David N. Chin, editors, *UMAP*, volume 6075 of *Lecture Notes in Computer Science*, pages 147–158. Springer, 2010.
17. A. Collins and A. Stevens. Goals and strategies for inquiry teachers. *Advances in Instructional Psychology*, 2:65–119, 1982.
18. Allan Collins, John Seely Brown, and Susan E. Newman. Cognitive apprenticeship: Teaching the craft of reading, writing and mathematics. In L. B. Resnick, editor, *Knowing, learning and instruction: Essays in honor of Robert Glaser*, chapter 14, pages 453–494. Lawrence Erlbaum Associates: Hillsdale New Jersey, 1989.
19. Christina Conati and Kurt VanLehn. Toward computer-based support of meta-cognitive skills: a computational framework to coach self-explanation. *International Journal of Artificial Intelligence in Education*, 11:398–415, 2000.
20. Albert T. Corbett and John R. Anderson. Locus of feedback control in computer-based tutoring: impact on learning rate, achievement and attitudes. In *CHI*, pages 245–252, 2001.
21. Sidney K. D'Mello, Scotty D. Craig, Amy M. Witherspoon, Bethany McDaniel, and Arthur C. Graesser. Automatic detection of learner's affect from conversational cues. *User Modeling and User-Adapted Interaction*, 18(1-2):45–80, 2008.
22. Sidney K. D'Mello and Arthur C. Graesser. Multimodal semi-automated affect detection from conversational cues, gross body language, and facial features. *User Modeling and User-Adapted Interaction*, 20(2):147–187, 2010.
23. Katherine Forbes-Riley, Diane J. Litman, Amruta Purandare, Mihai Rotaru, and Joel R. Tetreault. Comparing linguistic features for modeling learning in computer tutoring. In Luckin et al. [45], pages 270–277.
24. Matthew Frampton and Oliver Lemon. Reinforcement learning of dialogue strategies using the user's last dialogue act. In *Proceedings of the IJCAI Workshop on K&R in Practical Dialogue Systems*, pages 62–67, 2005.

25. Matthew Frampton and Oliver Lemon. Learning more effective dialogue strategies using limited dialogue move features. In Calzolari et al. [10], pages 185–192.
26. Arthur C. Graesser, Natalie K. Person, and Joseph P. Magliano. Collaborative dialog patterns in naturalistic one-on-one tutoring. *Applied Cognitive Psychology*, 9(6):495–422, December 1995.
27. Arthur C. Graesser, Kurt VanLehn, Carolyn Penstein Rosé, Pamela W. Jordan, and Derek Harter. Intelligent tutoring systems with conversational dialogue. *AI Magazine*, 22(4):39–52, 2001.
28. Milos Hauskrecht. *Planning and control in stochastic domains with imperfect information.* PhD thesis, MIT, 1997. Available as Technical Report: MIT-LCS-TR-738, 1997.
29. James Henderson, Oliver Lemon, and Kallirroi Georgila. Hybrid reinforcement/supervised learning for dialogue policies from communicator data. In *IJCAI Workshop on K&R in Practical Dialogue Systems*, pages 68–75, 2005.
30. Ana Iglesias, Paloma Martínez, Ricardo Aler, and Fernando Fernández. Learning teaching strategies in an adaptive and intelligent educational system through reinforcement learning. *Applied Intelligence*, 31:89–106, 2009. 10.1007/s10489-008-0115-1.
31. Ana Iglesias, Paloma Martínez, Ricardo Aler, and Fernando Fernández. Reinforcement learning of pedagogical policies in adaptive and intelligent educational systems. *Knowledge-Based Systems*, 22(4):266–270, 2009. Artificial Intelligence (AI) in Blended Learning - (AI) in Blended Learning.
32. Ana Iglesias, Paloma Martínez, and Fernando Fernández. An experience applying reinforcement learning in a web-based adaptive and intelligent educational system. *Informatics in Education*, 2(2):223–240, 2003.
33. Mitsuru Ikeda, Kevin D. Ashley, and Tak-Wai Chan, editors. *Intelligent Tutoring Systems, 8th International Conference, ITS 2006, Jhongli, Taiwan, June 26-30, 2006, Proceedings*, volume 4053 of *Lecture Notes in Computer Science.* Springer, 2006.
34. Srinivasan Janarthanam and Oliver Lemon. User simulations for online adaptation and knowledge-alignment in troubleshooting dialogue systems. In *Proceedings of LonDial the 12th SEMdial Workshop on on the Semantics and Pragmatics of Dialogues.*, pages 51–58, Stockholm, 2008.
35. I. T. Jolliffee. *Principal Component Analysis.* Springer Series in Statistics. Springer, New York, 2nd edition, 2002.
36. Pamela W. Jordan, Brian Hall, Michael Ringenberg, Yui Cue, and Carolyn Rosé. Tools for authoring a dialogue agent that participates in learning studies. In Luckin et al. [45], pages 43–50.
37. Pamela W. Jordan, Michael A. Ringenberg, and Brian Hall. Rapidly developing dialogue systems that support learning studies. In *ITS06 Workshop on Teaching with Robots, Agents and NLP, available http://facweb.cs.depaul.edu/elulis/ITS2006RobotsAgentsWorkshop.html*, pages 29–36, 2006.
38. Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
39. Sandra Katz, Gabriel O'Donnell, and Heater Kay. An approach to analyzing the role and structure of reflective dialogue. *International Journal of Artificial Intelligence and Education*, 11(3):320–343, 2000.

40. Kenneth R. Koedinger and Vincent Aleven. Exploring the assistance dilemma in experiments with cognitive tutors. *Educational Psychology Review*, 19(3):239–264, 2007.
41. Kenneth R. Koedinger, John R. Anderson, William H. Hadley, and Mary A. Mark. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8(1):30–43, 1997.
42. James C. Lester, Rosa Maria Vicari, and Fábio Paraguaçu, editors. *Intelligent Tutoring Systems, 7th International Conference, ITS 2004*, volume 3220 of *Lecture Notes in Computer Science*, Maceiò, Alagoas, Brazil, August 30-September 3 2004. Springer.
43. Esther Levin and Roberto Pieraccini. A stochastic model of computer-human interaction for learning dialogue strategies. In *In EUROSPEECH 97*, pages 1883–1886, 1997.
44. Diane J. Litman and Scott Silliman. Itspoke: an intelligent tutoring spoken dialogue system. In *Demonstration Papers at HLT-NAACL 2004*, pages 5–8, Morristown, NJ, USA, 2004. Association for Computational Linguistics.
45. Rosemary Luckin, Kenneth R. Koedinger, and Jim E. Greer, editors. *Artificial Intelligence in Education, Building Technology Rich Learning Contexts That Work, Proceedings of the 13th International Conference on Artificial Intelligence in Education, AIED 2007*, volume 158 of *Frontiers in Artificial Intelligence and Applications*, Los Angeles, California, USA, July 9-13 2007. IOS Press.
46. Kimberly N. Martin and Ivon Arroyo. Agentx: Using reinforcement learning to improve the effectiveness of intelligent tutoring systems. In Lester et al. [42], pages 564–572.
47. Jean McKendree. Effective feedback content for tutoring complex skills. *Human-Computer Interaction*, 5(4):381–413, December 1990.
48. Johanna D. Moore, Kaska Porayska-Pomsta, Sebastian Varges, and Claus Zinn. Generating tutorial feedback with affect. In Valerie Barr and Zdravko Markov, editors, *FLAIRS Conference*, pages 923–928. AAAI Press, 2004.
49. Allen Newell. *Unified Theories of Cognition.* Harvard University Press; Reprint edition, 1994.
50. Tim Paek and David Chickering. The Markov assumption in spoken dialogue management. In *6th SIGDial Workshop on Discourse and Dialogue*, pages 35–44, 2005.
51. Helen Pain and Kaska Porayska-Pomsta. Affect in one-to-one tutoring. In Ikeda et al. [33], pages 817–817.
52. Pipatsarun Phobun and Jiracha Vicheanpanya. Adaptive intelligent tutoring systems for e-learning systems. *Procedia - Social and Behavioral Sciences*, 2(2):4064 – 4069, 2010. Innovation and Creativity in Education.
53. Kaska Porayska-Pomsta, Manolis Mavrikis, and Helen Pain. Diagnosing and acting on student affect: the tutor's perspective. *User Modeling and User-Adapted Interaction*, 18(1-2):125–173, 2008.
54. Antoine Raux, Brian Langner, Dan Bohus, Alan W. Black, and Maxine Eskenazi. Let's go public! taking a spoken dialog system to the real world. In *Proceedings of Interspeech (Eurospeech)*, pages 885–888, Lisbon Portugal, 2005.
55. Verena Rieser and Oliver Lemon. Using machine learning to explore human multimodal clarification strategies. In Calzolari et al. [10], pages 659–666.
56. Michael A. Ringenberg and Kurt VanLehn. Scaffolding problem solving with annotated, worked-out examples to promote deep learning. In Ikeda et al. [33], pages 625–634.

57. A. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Shern, K. Lenzo, W. Xu, and A. Oh. Creating natural dialogs in the Carnegie Mellon communicator system. In *Proceedings of Eurospeech*, volume 4, pages 1531–1534, 1999.
58. Satinder P. Singh, Michael J. Kearns, Diane J. Litman, and Marilyn A. Walker. Reinforcement learning for spoken dialogue systems. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *NIPS*, pages 956–962. The MIT Press, 1999.
59. Satinder P. Singh, Diane J. Litman, Michael J. Kearns, and Marilyn A. Walker. Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *Journal of Aritificial Intelligence Research (JAIR)*, 16:105–133, 2002.
60. John C. Stamper, Tiffany Barnes, and Marvin J. Croy. Extracting student models for intelligent tutoring systems. In *AAAI*, pages 1900–1901, Vancouver, British Columbia, Canada, July 22-26 2007. AAAI Press.
61. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning.* MIT Press Bradford Books, 1998.
62. Joel R. Tetreault, Dan Bohus, and Diane J. Litman. Estimating the reliability of mdp policies: a confidence interval approach. In Candace L. Sidner, Tanja Schultz, Matthew Stone, and ChengXiang Zhai, editors, *HLT-NAACL*, pages 276–283. The Association for Computational Linguistics, 2007.
63. Joel R. Tetreault and Diane J. Litman. Comparing the utility of state features in spoken dialogue using reinforcement learning. In Robert C. Moore, Jeff A. Bilmes, Jennifer Chu-Carroll, and Mark Sanderson, editors, *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 272–279, New York, NY, USA, June 2006. The Association for Computational Linguistics.
64. Joel R. Tetreault and Diane J. Litman. Using reinforcement learning to build a better model of dialogue state. In *Proceedings 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 289–296, Trento, Italy, 2006.
65. Joel R. Tetreault and Diane J. Litman. A reinforcement learning approach to evaluating state representations in spoken dialogue systems. *Speech Communication*, 50(8-9):683–696, 2008.
66. Kurt VanLehn. The behavior of tutoring systems. *International Journal Artificial Intelligence in Education*, 16(3):227–265, 2006.
67. Kurt VanLehn, Arthur C. Graesser, G. Tanner Jackson, Pamela W. Jordan, Andrew Olney, and Carolyn Penstein Rosé. When are tutorial dialogues more effective than reading? *Cognitive Science*, 31(1):3–62, 2007.
68. Kurt VanLehn, Pamela Jordan, and Diane Litman. Developing pedagogically effective tutorial dialogue tactics: Experiments and a testbed. In *Proceedings of SLaTE Workshop on Speech and Language Technology in Education ISCA Tutorial and Research Workshop*, pages 17–20, 2007.
69. Kurt VanLehn, Collin Lynch, Kay Schulze, Joel A. Shapiro, Robert Shelby, Linwood Taylor, Don Treacy, Anders Weinstein, and Mary Wintersgill. The andes physics tutoring system: Lessons learned. *International Journal Artificial Intelligence in Education*, 15(3):147–204, 2005.
70. Lev S. Vygotsky. Interaction between learning and development. In *Mind and Society*, pages 79–91. Harvard University Press, Cambridge Massachusetts, 1978.

71. Marilyn A. Walker. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *Journal of Aritificial Intelligence Research*, 12:387–416, 2000.

72. Jason D. Williams, Pascal Poupart, and Steve J. Young. Factored partially observable markov decision processes for dialogue management. In *4th Workshop on Knowledge and Reasoning in Practical Dialog Systems, International Joint Conference on Artifiical Intelligence (IJCAI)*, pages 76–82, Edinburgh, 2005.

73. Jason D. Williams and Steve Young. Partially observable markov decision processes for spoken dialog systems. *Computer Speech and Language*, 21(2):231–422, April 2007.

74. Jason D. Williams and Steve Young. Scaling POMDPs for spoken dialog management. *IEEE Transactions on Audio, Speech, and Language Processing.*, 15(7):2116–2129, September 2007.

75. Ruth Wylie, Kenneth Koedinger, and Teruko Mitamura. Is self-explanation always better? the effects of adding self-explanation prompts to an english grammar tutor. In *Proceedings of the 31st Annual Conference of the Cognitive Science Society, COGSCI 2009*, pages 1300–1305, Amsterdam, The Netherlands, 2009.