When to Intervene: Toward a Markov Decision Process Dialogue Policy for Computer Science Tutoring

Christopher M. Mitchell, Kristy Elizabeth Boyer, and James C. Lester

Department of Computer Science, North Carolina State University,
Raleigh, North Carolina, USA
{cmmitch2, keboyer, lester}@ncsu.edu

Abstract. Designing dialogue systems that engage in rich tutorial dialogue has long been a goal of the intelligent tutoring systems community. A key challenge for these systems is determining when to intervene during student problem solving. Although intervention strategies have historically been hand-authored, utilizing machine learning to automatically acquire corpus-based intervention policies that maximize student learning holds great promise. To this end, this paper presents a Markov Decision Process (MDP) framework to learn when to intervene, capturing the most effective tutor turn-taking behaviors in a task-oriented learning environment with textual dialogue. This framework is developed as a part of the JavaTutor tutorial dialogue project and will contribute to data-driven development of a tutorial dialogue system for introductory computer science education.

Keywords: Tutorial Dialogue, Markov Decision Processes, Reinforcement Learning

1 Introduction

The effectiveness of tutorial dialogue has been widely established [1, 2]. Today's tutorial dialogue systems have been successful in producing learning gains as they support problem solving [3–5], encourage collaboration [6, 7], and adapt to student responses [8]. These systems have also been shown to be successful in implementing some affective adaptations of human tutors [5, 9]. Recent research into tutorial dialogue systems with unrestricted turn-taking has shown promise for simulating the natural tutorial dialogue interactions of a human tutor [7]. Recognizing and simulating the natural conversational turn-taking behavior of humans continues to be an area of active research [10–12], and there has recently been renewed interest in developing dialogue systems that harness unrestricted turn-taking paradigms [7, 13, 14].

The JavaTutor tutorial dialogue project aims to build a tutorial dialogue system with unrestricted turn-taking and rich natural language to support introductory computer science students. The overarching paradigm of this project is to automatically derive tutoring strategies using machine learning techniques applied to a corpus collected from an observational study of human-human tutoring. In

particular, the project focuses on how to devise tutorial strategies that deliver both cognitive and affective scaffolding in the most effective way. The project to date has seen the collection of a large corpus of tutorial dialogue featuring six repeated interactions with tutor-student pairs, accompanied by data on learning and attitude for each session as well as across the study [15–17]. This paper describes an important first step toward deriving tutorial dialogue policies automatically from the collected corpus in a way that does not simply mimic the behavior of human tutors, but seeks to identify the most effective tutorial strategies and implement those within the system's dialogue policy.

In recent years, reinforcement learning (RL) has proven useful for creating tutorial dialogue system policies in structured problem-solving interactions, such as what type of question to ask a student [18] and whether to elicit or tell the next step in the solution [19]. In order to harness the power of RL-based approaches within a tutorial dialogue system for computer science education, two important research problems must be addressed. First, a representation must be formulated in which student computer programming actions, which can occur continuously or in small bursts, can be segmented at an appropriate granularity and provided to the model. Second, because student dialogue moves, tutor dialogue moves, and student programming actions can occur in an interleaved manner with some overlapping each other, features to define the Markov Decision Process state space must be identified that preserve the rich, unrestricted turn-taking and mixed-initiative interaction to the greatest extent possible. In a first effort to address these challenges, this paper presents a novel application of RL-based approaches to the JavaTutor corpus of textual tutorial dialogue. In particular, the focus here is automatically learning when to intervene from this fixed corpus of human-human task-oriented tutorial dialogue with unrestricted turn-taking. The presented approach and policy results can inform data-driven development of tutorial systems for computer science education.

2 Human-Human Tutorial Dialogue Corpus

To date, the JavaTutor project has seen the collection of an extensive corpus of human-human tutoring. Between August 2011 and March 2012, 67 students interacted with experienced tutors through the Java Online Tutoring Environment (Figure 1). Students were drawn from a first-year engineering course on a voluntary basis. They earned partial course credit for their participation. Students who reported substantial programming experience in a pre-survey were excluded from the experienced-tutoring condition (and were instead placed in a peer-tutoring collaborative condition that is beyond the scope of this paper), since the target population of the JavaTutor tutorial dialogue system is students with no programming experience. Each student completed six tutoring sessions over a period of four weeks, and worked with the same tutor for all interactions. Each tutoring session was limited to forty minutes.

Seven tutors participated in the study. Their experience level ranged from multiple years' experience in one-on-one tutoring to one semester's experience as a teaching assistant or small group tutor. Gender distribution of the tutors was three female and

four male. Tutors were provided with printed learning objectives for each session and were reminded that they should seek to support the students' learning as well as motivational and emotional state. Also, because each subsequent tutoring session built on the completed computer program from the preceding session, tutors were encouraged to ensure that students completed the required components of the programming task within the allotted forty-minute time frame.

The overarching computer science problem-solving task was for students to create a text-based adventure game in which a player can explore scenes based on menu choices. In order to implement the adventure game, students learned a variety of programming concepts and constructs. This paper focuses on the first of the six tutoring sessions. The learning objectives covered in this first session included compiling and running code, writing comments, variable declaration, and system I/O. For each learning objective, there was a conceptual component and an applied component. For example, for the learning objective related to compiling code, the conceptual learning objective was for students to explain that compilation translates human-readable Java programs into machine-readable forms. The applied learning objective was for students to demonstrate that they can compile a program by pressing the "compile" button within the interface.

The Java Online Learning Environment, shown in Figure 1, supports textual dialogue between the human tutor and student. It also provides tutors with a real-time synchronized view of the student's workspace. The interface allows for logging events to a database with millisecond precision, making it straightforward to reconstruct the events of a session from these logs. There are two information channels between a tutor and a student. The first of these, the messaging pane, supports unrestricted textual dialogue between a tutor and a student, similar to common instant messaging applications. There are no restrictions placed on turn-taking, allowing either person to compose a message at any time. In addition, both students and tutors are notified when their partner is composing a message. The second information channel is the student's workspace. A tutor can see progress on the Java program written by the student in real-time, but the tutor is not able to edit the program directly. The Java programming environment is scaffolded for novices: it hides class declarations, method declarations, and import statements from the student, lowering the amount of complex syntax visible. Students effectively compose their programs within a main method "sandbox".

In order to measure the effectiveness of each session, students completed a pre-test at the beginning of each session and a post-test at the end of each session evaluating their knowledge of the material to be taught in that lesson. From these, we computed normalized learning gain using the following equation:

$$normalizedLearningGain = \begin{cases} \dfrac{posttest - pretest}{1 - pretest}, & posttest > pretest \\ \dfrac{posttest - pretest}{pretest}, & posttest \leq pretest \end{cases} \quad (1)$$

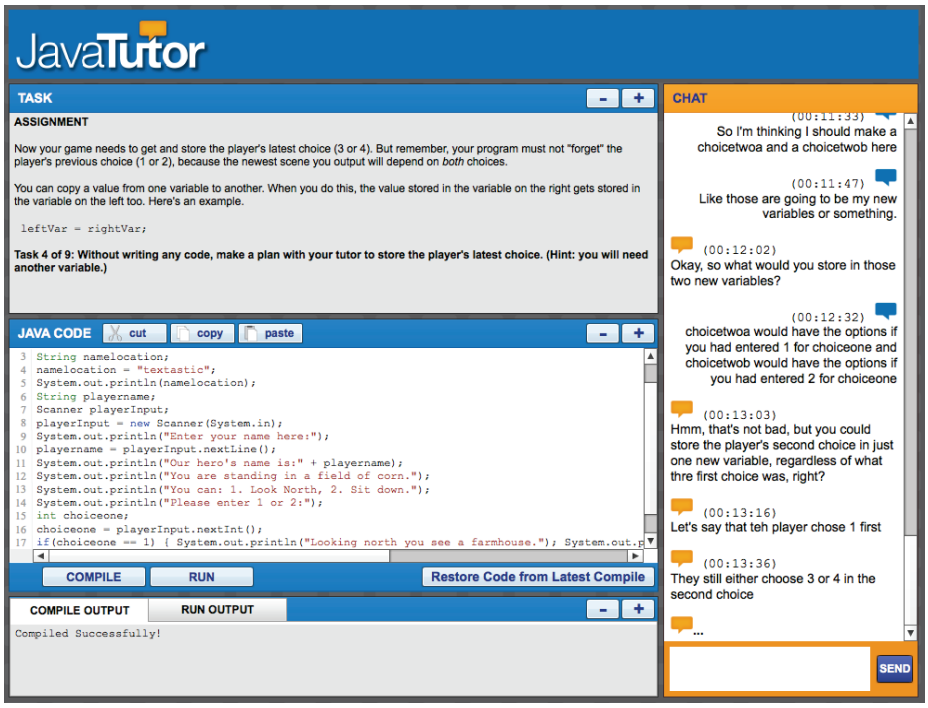

Fig. 1. A student's view of the JavaTutor human tutoring interface

This equation, adapted from Marx and Cummings [20] allows for the possibility of negative learning gain during a session, a phenomenon that occurred three times in the corpus. These normalized learning gain values can range from -1 to 1. In the present study normalized learning gains ranged from -0.29 to 1 (mean = 0.42; median = 0.45; st. dev. = 0.32). Students scored significantly higher on the post-test than the pre-test ($p < .001$).

3 Building the Markov Decision Process

The goal of the analysis presented here is to derive an effective tutorial intervention policy—*when* to intervene—from a fixed corpus of student-tutor interactions. From the tutors' perspective, the decision to intervene was made based on the state of the interaction as observed through the two information channels in the interface: the textual dialogue pane and the synchronized view of the student's workspace. In order to use a MDP framework to derive an effective intervention policy, we describe a representation of the interaction state as a collection of features from these information channels.

A Markov Decision Process is a model of a system in which a policy can be learned to maximize reward [21]. It consists of a set of states S, a set of actions A representing possible actions by an agent, a set of transition probabilities indicating

how likely it is for the model to transition to each state $s' \in S$ from each state $s \in S$ when the agent performs each action $a \in A$ in state s, and a reward function R that maps real values onto transitions and/or states, thus signifying their utility.

The goal of this analysis is to model tutor interventions during the task-completion process, so the possible actions for a tutor were to intervene (by composing and sending a message) or not to intervene. Hence, the set of actions is defined as $A = \{TutorMove, NoMove\}$. We chose three features to represent the state of the dialogue, with each feature taking on one of three possible values. These features, described in Table 1, combine as a triple to form the states of the MDP as (Current Student Action, Task Trajectory, Last Action). These three features were chosen because they succinctly represent the current state of the dialogue in terms of turn-taking information in the *Current Action* and *Last Action* features, while the recent behavior of the student is captured in the *Task Trajectory* and *Current Action* features. Thus, these features supply an agent with sufficient information to learn a basic intervention policy while relying only on automatically annotated features. By selecting a small state space and action space, we avoid data sparsity issues [22], thereby decreasing the likelihood of states being insufficiently explored in our corpus, and increasing the likelihood of producing a meaningful intervention policy.

Table 1. The features that define the states of the Markov Decision Process

Current Student Action	Task Trajectory	Last Action
• ***Task***: Working on the task • ***StudentDial***: Writing a message to the tutor • ***NoAction***: No current student action	• ***Closer***: Moving closer to the final correct solution • ***Farther***: Moving away from correct solution • ***NoChange***: Same distance from correct solution	• ***TutorDial***: Tutor message • ***StudentDial***: Student message • ***Task*:** Student worked on the task

In addition, the model includes 3 more states: an *Initial* state, in which the model always begins, and two final states: one with reward +100 for students achieving higher-than-median normalized learning gain and one with reward -100 for the remaining students, following the conventions established in prior research into reinforcement learning for tutorial dialogue [18, 19].

Using these formalizations, one state was assigned to each of the log entries collected during the sessions and transition probabilities were computed between them when a tutor made an intervention (*TutorMove*) and when a tutor did not make an intervention (*NoMove*) based on the transition frequencies observed in the data. Any states that occurred less than once per session on average were combined into a single *LowFrequency* state, following the convention of prior work [23]. There were four states fitting this description: (*Task*, *Farther*, *StudentDial*), (*StudentDial*, *Farther*, *StudentDial*), (*StudentDial*, *Farther*, *Task*), and (*StudentDial*, *Farther*, *TutorDial*). Thus, the final MDP model contained 25 states requiring a tutorial intervention decision (23 states composed of feature combinations, the *LowFrequency* state, and the *Initial* state), and two final states.

The *Current Student Action* and *Last Action* features were relatively straightforward to assign to log entries by simply observing what a student was currently doing at that point in the session and observing what action had occurred most recently. The *Task Trajectory* feature was computed by discretizing the students' work on the task into chunks, which presents a substantial research question and design decision for supporting computer science learning. Historically, intelligent tutoring systems for computer science have utilized granularity at one extreme or the other. The smallest possible granularity is every keystroke, perhaps the earliest example of this being the Lisp tutor of Anderson and colleagues [24]. The largest granularity could arguably be to evaluate only when the student deems the artifact complete enough to manually submit for evaluation, which was the approach taken by another very early computer science tutor, Proust [25]. For the JavaTutor system, evaluating the student program more often than at the completion of tasks is essential to support dialogue, but an every-keystroke evaluation is too frequent due in part to algorithm runtime limitations. We define our task events as beginning when a student begins typing in the task pane and ending when a student has not typed in the task pane for at least 1.5 seconds. This threshold of 1.5 seconds was chosen empirically before model building to strike a balance between shorter thresholds, which resulted in frequent switching between "working on task" and "not working on task" states, and longer thresholds, which resulted in never leaving the "working on task" state.

After each task event (discretized as described above), a student's program was separated into tokens as defined by the Java compiler, and a token-level minimum edit distance was computed from that student's final solution for the lesson, tokenized in the same manner. Variable names, comments, and the contents of string literals were ignored in this edit distance calculation. The change in the edit distance from one chunk to the next determined the value of the *Task Trajectory* feature. Because the tutors were experienced in Java programming and had knowledge of the lesson structure, it is reasonable to assume that they were able to determine whether the student was moving farther or closer to the final solution. In this way, the edit distance algorithm provides a rough, automatically computable estimate of the tutors' assessment of student progress.

4 Policy Learning

The goal of this analysis is to learn a tutorial intervention policy—*when* to intervene—that reflects the most effective strategies within the corpus. In the MDP framework described above, this involves maximizing the learning gain reward. In order to learn this tutorial intervention policy, we used a policy iteration algorithm [21] on the MDP. For each iteration, this algorithm computes the expected reward in each state $s \in S$ when taking each action $a \in A$, based on the computed transition probabilities to other states and the expected rewards of those states from the previous iteration. Following the practice of prior work [13, 17], a discount factor of 0.9 was used to penalize delayed rewards (those requiring several state transitions to achieve) in favor of immediate rewards (those requiring few state transitions to achieve). The

policy iteration continues until convergence is reached; that is, until the change in expected reward for each state is less than some epsilon value between iterations. We used an epsilon of 10^{-7}, requiring 125 iterations to converge. The resulting policy is shown in Table 2.

Table 2. The learned tutorial intervention policy

State (*Current Action*, *Task Trajectory*, *Last Action*)	**Policy**
(Task, Closer, Task)	TutorMove
(Task, Closer, StudentDial)	TutorMove
(Task, Closer, TutorDial)	TutorMove
(Task, Farther, Task)	TutorMove
(Task, Farther, TutorDial)	TutorMove
(Task, NoChange, Task)	TutorMove
(Task, NoChange, StudentDial)	NoMove
(Task, NoChange, TutorDial)	TutorMove
(StudentDial, Closer, Task)	TutorMove
(StudentDial, Closer, StudentDial)	TutorMove
(StudentDial, Closer, TutorDial)	TutorMove
(StudentDial, NoChange, Task)	NoMove
(StudentDial, NoChange, StudentDial)	NoMove
(StudentDial, NoChange, TutorDial)	NoMove
(NoAction, Closer, Task)	TutorMove
(NoAction, Closer, StudentDial)	TutorMove
(NoAction, Closer, TutorDial)	NoMove
(NoAction, Farther, Task)	NoMove
(NoAction, Farther, StudentDial)	TutorMove
(NoAction, Farther, TutorDial)	NoMove
(NoAction, NoChange, Task)	TutorMove
(NoAction, NoChange, StudentDial)	NoMove
(NoAction, NoChange, TutorDial)	NoMove
Initial	TutorMove
LowFrequency	TutorMove

Some noteworthy patterns emerge in the intervention policy learned from the corpus. For example, in seven of the eight states where the student is actively engaged in task actions (*Task*, *, *), the policy recommends that the tutor make a dialogue move. An excerpt from the corpus illustrating this strategy in a high learning gain session is shown in Figure 2, on lines 2-4. An excerpt from a low learning gain session showing tutor non-intervention during task progress is shown in Figure 3. In addition, among the states in which no action is currently being taken by the student and the last action was a tutor message, i.e., matching the pattern (*NoAction*, *, *TutorDial*), we find that the policy recommends that a tutor not make another consecutive dialogue move, regardless of how well the student is progressing on the task. However, Figure 2 shows that high learning gains are possible without strictly following this particular recommendation. Additional discussion on these recommendations can be found in [26].

5 Conclusion and Future Work

Current tutorial dialogue systems are highly effective, and matching the effectiveness of the most effective tutors is a driving force of tutorial dialogue research. This paper

presents a step toward rich, adaptive dialogue for supporting computer science learning by introducing a representation of task-oriented dialogue with unrestricted turn-taking in a reinforcement learning framework and presenting initial results of an automatically learned policy for when to intervene. The presented approach will inform the development of the JavaTutor tutorial dialogue system, whose initial policies will be learned based on the fixed human-human corpus described here.

Event	Tutor action and state transition
1. *Student is declaring a String variable named "aStringVariable".*	*NoMove* ↓ (Task, NoChange, Task)
2. *Tutor starts typing a message*	*TutorMove*
3. *1.5 seconds elapse, task action is complete.*	↓
4. **Tutor message:** That works, but let's give the variable a more descriptive name	(NoAction, Closer, TutorDial)
5. *Tutor starts typing a message*	*TutorMove*
6. *Student starts typing a message*	↓
7. **Student message:** ok	
8. **Tutor message:** Usually, the variable's name tells us what data it has stored	(NoAction, Closer, TutorDial)

Fig. 2. An excerpt from a high learning gain session.

Event	Tutor action and state transition
1. *Student has just attempted to implement the programming code needed to complete the task, with no tutor intervention.*	*NoMove* ↓ (NoAction, Closer, Task)
2. *Student starts typing a message*	*NoMove* ↓ (StudentDial, Closer, Task)
3. **Student message:** not sure if this is right…	*NoMove* ↓ (NoAction, Closer, StudentDial)

Fig. 3. An excerpt from a low learning gain session.

Further exploring of the state space via simulation and utilizing a more expressive representation of state are highly promising directions for future work. Other directions for future work include undertaking a more fine-grained analysis of the timing of interventions, which could inform the development of more natural interactions, as well as allowing for more nuanced intervention strategies. Additionally, these models should be enhanced with a more expressive representation of both dialogue and task. It is hoped that these lines of investigation will yield highly effective machine-learned policies for tutorial dialogue systems and that tutorial dialogue systems for computer science will make this subject more accessible to students of all grade levels.

Acknowledgements

The JavaTutor project team includes Eric Wiebe, Bradford Mott, Eun Young Ha, Joseph Grafsgaard, Alok Baikadi, Megan Hardy, Mary Luong, Miles Smaxwell, Natalie Kerby, Robert Fulton, Caitlin Foster, Joseph Wiggins, and Denae Ford. This work is supported in part by the National Science Foundation through Grants DRL-1007962 and CNS-1042468. Any opinions, findings, conclusions, or recommendations expressed in this report are those of the participants, and do not necessarily represent the official views, opinions, or policy of the National Science Foundation.

References

1. Bloom, B.: The 2 sigma problem: the search for methods of group instruction as effective as one-to-one tutoring. Educational Researcher. 13, 4–16 (1984).
2. VanLehn, K., Graesser, A.C., Jackson, G.T., Jordan, P., Olney, A., Rosé, C.P.: When are tutorial dialogues more effective than reading? Cognitive Science. 31, 3–62 (2007).
3. Evens, M.W., Michael, J.: One-on-One Tutoring by Humans and Computers. Lawrence Erlbaum Associates, Mahwah, New Jersey (2005).
4. Heffernan, N.T., Koedinger, K.: The design and formative analysis of a dialog-based tutor. Workshop on Tutorial Dialogue Systems. pp. 23–34 (2001).
5. Forbes-Riley, K., Litman, D.: Adapting to student uncertainty improves tutoring dialogues. Proceedings of the International Conference on Artificial Intelligence in Education. pp. 33–40 (2009).
6. Kersey, C., Di Eugenio, B., Jordan, P., Katz, S.: KSC-PaL: A peer learning agent that encourages students to take the initiative. Proceedings of the Fourth Workshop on Innovative Use of NLP for Building Educational Applications. pp. 55–63 (2009).
7. Kumar, R., Rosé, C.P.: Architecture for Building Conversational Agents that Support Collaborative Learning. IEEE Transactions on Learning. 4, 21–34 (2011).
8. Jackson, G.T., Person, N.K., Graesser, A.C.: Adaptive Tutorial Dialogue in AutoTutor. ITS 2004 Workshop Proceedings on Dialog-based Intelligent Tutoring Systems. pp. 9–13 (2004).
9. D'Mello, S., Graesser, A.: AutoTutor and affective autotutor: Learning by talking with cognitively and emotionally intelligent computers that talk back. ACM Transactions on Interactive Intelligent Systems. 2, (2012).
10. Jonsdottir, G.R., Thorisson, K.R., Nivel, E.: Learning Smooth, Human-Like Turntaking in Realtime Dialogue. Proceedings of the 8th International Conference on Intelligent Virtual Agents. pp. 162–175 (2008).
11. Ward, N.G., Fuentes, O., Vega, A.: Dialog Prediction for a General Model of Turn-Taking. Proceedings of the International Conference on Spoken Language Processing (2010).

12. Raux, A., Eskenazi, M.: Optimizing the turn-taking behavior of task-oriented spoken dialog systems. ACM Transactions on Speech and Language Processing. 9, 1–23 (2012).
13. Bohus, D., Horvitz, E.: Multiparty Turn Taking in Situated Dialog: Study, Lessons, and Directions. Proceedings of the 12th Annual Meeting of the Special Interest Group in Discourse and Dialogue. pp. 98–109 (2011).
14. Morbini, F., Forbell, E., DeVault, D., Sagae, K., Traum, D.R., Rizzo, A.A.: A Mixed-Initiative Conversational Dialogue System for Healthcare. Proceedings of the 13th Annual Meeting of the Special Interest Group in Discourse and Dialogue. pp. 137–139 (2012).
15. Mitchell, C.M., Boyer, K.E., Lester, J.C.: From strangers to partners: examining convergence within a longitudinal study of task-oriented dialogue. Proceedings of the 13th Annual SIGDIAL Meeting on Discourse and Dialogue. pp. 94–98 (2012).
16. Ha, E.Y., Grafsgaard, J.F., Mitchell, C.M., Boyer, K.E., Lester, J.C.: Combining verbal and nonverbal features to overcome the "information gap" in task-oriented dialogue. Proceedings of the 13th Annual SIGDIAL Meeting on Discourse and Dialogue. pp. 246–256 (2012).
17. Grafsgaard, J.F., Fulton, R., Boyer, K.E., Wiebe, E., Lester, J.C.: Multimodal analysis of the implicit affective channel in computer-mediated textual communication. to appear in Proceedings of the 14th ACM international conference on Multimodal Interaction (2012).
18. Tetreault, J.R., Litman, D.J.: A Reinforcement Learning approach to evaluating state representations in spoken dialogue systems. Speech Communication. 50, 683–696 (2008).
19. Chi, M., VanLehn, K., Litman, D.: Do micro-level tutorial decisions matter: applying reinforcement learning to induce pedagogical tutorial tactics. Proceedings of the International Conference on Intelligent Tutoring Systems. pp. 224–234. (2010).
20. Marx, J.D., Cummings, K.: Normalized change. American Journal of Physics. 75, 87–91 (2007).
21. Sutton, R., Barto, A.: Reinforcement Learning. MIT Press, Cambridge, MA (1998).
22. Singh, S., Litman, D., Kearns, M., Walker, M.: Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. Journal of Artificial Intelligence Research. 16, 105–133 (2002).
23. Tetreault, J.R., Litman, D.J.: Using Reinforcement Learning to Build a Better Model of Dialogue State. Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics. pp. 289–296 (2006).
24. Anderson, J.R., Boyle, C.F., Corbett, A.T., Lewis, M.W.: Cognitive modeling and intelligent tutoring. Artificial Intelligence. 42, 7–49 (1990).
25. Johnson, W.L., Soloway, E.: PROUST: Knowledge-based program understanding. ICSE '84: Proceedings of the 7th international conference on Software engineering. pp. 369–380 (1984).
26. Mitchell, C.M., Boyer, K.E., Lester, J.C.: A Markov Decision Process Model of Tutorial Intervention in Task-Oriented Dialogue. To appear in Proceedings of the 16th International Conference on Artificial Intelligence in Education (2013).