Evaluating State Representations for Reinforcement Learning of Turn-Taking Policies in Tutorial Dialogue

Christopher M. Mitchell **Kristy Elizabeth Boyer** **James C. Lester**
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
{cmmitch2, keboyer, lester}@ncsu.edu

Abstract

Learning and improving natural turn-taking behaviors for dialogue systems is a topic of growing importance. In task-oriented dialogue where the user can engage in task actions in parallel with dialogue, unrestricted turn taking may be particularly important for dialogue success. This paper presents a novel Markov Decision Process (MDP) representation of dialogue with unrestricted turn taking and a parallel task stream in order to automatically learn effective turn-taking policies for a tutorial dialogue system from a corpus. It also presents and evaluates an approach to automatically selecting features for an MDP state representation of this dialogue. The results suggest that the MDP formulation and the feature selection framework hold promise for learning effective turn-taking policies in task-oriented dialogue systems.

1 Introduction

Determining when to make a dialogue move is a topic of growing importance in dialogue systems. While systems historically relied on explicit turn-taking cues, more recent work has focused on learning and improving on natural turn-taking behaviors (Raux and Eskenazi 2012; Selfridge et al. 2012). For tutorial dialogue in particular, effectively timing system moves can substantially impact the success of the dialogue. For example, failing to provide helpful feedback to a student who is confused may lead to decreased learning (Shute 2008) or to disengagement (Forbes-Riley and Litman 2012), while providing tutorial feedback or interventions at inappropriate times could also have a negative impact on the outcome of the dialogue (D'Mello et al. 2010).

Reinforcement Learning (RL) is a widely used approach to constructing effective dialogue policies using either MDPs or POMDPs (Williams and Young 2007). To date, RL has been applied to learn the most effective dialogue move to make, but has not been applied to learning the timings of these moves, although the related concept of when to release a turn has been explored (English and Heeman 2005). The domain of tutorial dialogue poses an additional modeling challenge: the dialogue is task-oriented, but unlike many task-oriented dialogues in which all information is communicated via dialogue, students solve problems within a separate task stream which conveys essential information for dialogue management decisions.

This paper addresses dialogue with both unrestricted turn taking and a parallel task stream with a novel Markov Decision Process representation. Because turn boundaries are not clearly defined or enforced, we apply RL to the problem of *when* to make a dialogue move, rather than *what type* of dialogue move to make. In order to determine which criteria are most relevant to making this decision, the approach utilizes a feature selection approach based on a new *Separation Ratio* metric and compares the selected features against an existing approach based on expected cumulative reward (Chi et al. 2011). Finally, the resulting feature spaces are evaluated with simulated users acquired in a supervised fashion from held-out portions of the corpus. The results inform the development of turn-taking policies in task-oriented dialogue systems.

2 Corpus

The corpus used for this work was collected during 2011 and 2012 as part of the JavaTutor tutorial dialogue project. It consists of 66 textual dialogues between human tutors and students, with an average of 90 tutor dialogue moves and 36 student dialogue moves. Each pair interacted for through a computer-mediated interface to complete introductory computer programming tasks. Students edited their computer programs within a parallel task stream also collected as part of the corpus (see Appendix A). Tutors viewed the task actions synchronously through the interface. The success of each dialogue was measured by learning gain between pretest and posttest. Overall the dialogues were effective; the average learning gain was 42.3% (statistically > 0; $p < .0001$). The substantial variation in learning gains (*min*=-28.6%; *max*= 100%) will be leveraged within the MDP reward structure.

3 MDP Representation

A Markov Decision Process (MDP) models a system in which a policy can be learned to maximize reward (Sutton and Barto 1998). 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.

Previous applications of RL to dialogue systems, using both MDPs and POMDPs, have dealt with the decision of *what type* of dialogue move to make (Chi et al. 2011; Williams and Young 2007). These systems make this decision either at predetermined decision points (Tetreault and Litman 2008), following the trigger of a silence threshold (Raux and Eskenazi 2012), or when the system determines it has enough information to advance the dialogue (Selfridge et al. 2012). For the JavaTutor corpus, however, the tutor could choose to make a move at any time. Rather than applying handcrafted rules to determine decision points, we apply RL to learn *when* to make a dialogue move in order to maximize the success of the dialogue. For this MDP, the action set is defined as $A = \{TutorMove, NoMove\}$.

The states for the MDP consist of combinations of features representing the current state of the session. The possible features available for selection are described in Table 1, and are all automatically extracted from system logs. The *Task Trajectory* and *Edit Distance* features are based on computing a token-level edit distance from a student's program with respect to that student's final correct solution. This distance measures a student's progress over the course of a dialogue while avoiding the need to manually annotate the task stream. In a deployed system, this edit distance can be estimated by comparing to previously acquired solutions from other students.

Feature	Description	Values
Current Action	The current action being taken by the student	• TASK • STUDENTDIAL • NOACTION
Task Trajectory	The effect of the last task action on the edit distance to the final task solution	• CLOSER • FARTHER • NOCHANGE
Last Action	Last turn taken by either interlocutor	• TUTORDIAL • STUDENTDIAL • TASK
Number of Tutor Moves	Number of tutor turns taken thus far in the dialogue	• LOW (< 30) • MID (30-59) • HIGH (> 60)
Edit Distance	The edit distance to the final solution	• LOW (< 20) • MID (20-49) • HIGH (> 50)
Elapsed Idle Time	The number of seconds since the last student action	• LOW (< 7) • MID (7-15) • HIGH (> 15)

Table 1. Features available to be selected

Tutor moves are encoded as MDP actions, while student actions are encoded as transitions to a new state with a *NoMove* tutor action. To account for the possibility that both interlocutors could construct messages simultaneously or that dialogue and task actions could happen at the same time, the following protocol was applied: if a tutor was making a dialogue move (*i.e.*, typing a message), the state transition accompanying a student action was made after the tutor move was complete, and the student move was associated with that *TutorMove* action.

Another important consideration for this representation was how to segment the task stream into discrete actions. Through empirical investigation the timeout threshold of 1.5 seconds was selected as a balance between large numbers of successive task events or very few, most of which overlapped with tutor turns.

There were three additional states in the MDP: the *Initial* state and two final states, *FinalHigh* and *FinalLow*, occurring only at the end of a dialogue and providing rewards of +100 and −100, respectively. A median split on student learning gains was used to assign each dialogue to either the *FinalHigh* state or *FinalLow* state.

4 Feature Selection

While retaining all six features would allow for a rich state representation, it would also lead to issues with sparsity (Singh et al. 2002). In fact, nearly 90% of states averaged less than one visit per dialogue when using all six features, leading to inadequate coverage of the state space on which to build reliable MDP policies. This section compares two methods used to select features from among the six available.

The first approach is based on the *Expected Cumulative Reward* (ECR) in the initial state, a metric previously used to evaluate state representations for a tutorial dialogue system using RL (Chi et al. 2011; Tetreault and Litman 2008). A higher initial-state ECR indicates a higher probability of achieving a favorable outcome when following a reward-maximizing policy. Maximizing ECR has also been the focus of other feature selection approaches for RL (Misu and Kashioka 2012, Li et al. 2009).

While initial-state ECR provides a measure of the likelihood of a favorable outcome, it does not address how well a particular state representation captures key decision points. That is, it does not directly represent the extent to which each decision along the path to a successful outcome contributed to that outcome, or whether the second-best decision in a particular state would have been equally useful. In order to measure this difference, we introduce the *Separation Ratio* (SR), which represents how much better a particular policy is compared to its alternatives. SR for a state is calculated by taking the absolute difference between the estimated values of two actions in that state and dividing by the mean of the two values. SR for a policy is the mean of the SRs across all states.

An SR near zero for a state indicates that the decision to take one action over another in that state is likely to have little effect on the final outcome of the dialogue. On the other hand, a high SR indicates a crucial decision point, where taking an off-policy action leads to a much lower probability of a successful outcome. The intuition behind this metric is that a state representation that supports policies with high SR highlights features that are useful in executing an effective turn-taking policy, while a state representation that produces policies with low SR fails to capture this information.

Using these two metrics, we evaluated the utility of each of the six features. Starting with two empty state representations, one for each metric, a greedy algorithm added one feature at a time to each. That is, at each step for each metric, the feature was added that led to the highest value on the metric when combined with the features already chosen. For each of the two metrics, we built a state representation and used it as the basis for an MDP. This MDP was then trained with policy iteration (Sutton and Barto 1998), and the two state representations that led to the highest value on each metric were carried over to the next iteration. The goal here is to evaluate the relative utility of each feature, so we continued adding features until they were exhausted, leading to a full ordering of features for each condition (Table 2).

Iteration	**Initial-State ECR Feature Ordering**	**Mean SR Feature Ordering**
1	Last Action	Number of Tutor Moves
2	Task Trajectory	Edit Distance
3	Current Action	Last Action
4	Elapsed Idle Time	Current Action
5	Number of Tutor Moves	Elapsed Idle Time
6	Edit Distance	Task Trajectory

Table 2. Feature selection using Expected Cumulative Reward (ECR) and Separation Ratio (SR)

Given the orderings in Table 2, the next step in building a RL system is to decide which iteration of the feature spaces to use. That is, how does a system designer determine when to stop adding features? Previous work (Chi et al. 2011; Tetreault and Litman 2008) viewed an absolute increase in the value of initial-state ECR as a signal for the quality of a newly added feature. So, one could say that feature addition should stop if initial-state ECR does not increase between iterations. In the current analysis, however, this would result in termination at the second iteration for the mean SR ordering and termination at the first iteration for the initial-state ECR ordering. These undesirably early terminations most likely occur because the first features selected in both orderings represent tutor actions: a tutor can always choose to make a move, thus setting the *Last Action* feature to TUTORDIAL, and a tutor has direct control over the value of *Number of Tutor Moves*. This control of features leads to deterministic control of state if the context provided by student-driven features is absent. This can allow a policy to remain in the state that maximizes the transition probability to the end state, thus increasing ECR for all states due to deterministic transitions. Therefore, a different type of stopping criterion is required.

A stopping criterion must balance two competing goals. On the one hand, the size of the state space must be limited to avoid issues with sparsity, as state-action pairs that are not well explored during training might not be assigned values proportional to their expected rewards in a deployed system. On the other hand, a feature space that is too small may not sufficiently represent the possible states of the world, and might fail to capture the criteria most relevant to making decisions. These competing goals of compactness and descriptive power must both be considered when choosing an appropriate feature space for a RL model.

In an attempt to balance these goals, we propose a stopping criterion based on the ratio of states that are sparse states. A sparse state is defined as any state that occurs less than once per dialogue on average. A sharp increase in sparse states was observed between the third and fourth iterations for both metrics (15% to 56% for ECR and 26% to 47% for SR), so feature addition stopped at the third iteration. This resulted in only one of the three selected features being shared among the two conditions: the *Last Action* made by either person (Table 2). In addition, both feature sets include a feature related to the task progress of the student: *Task Trajectory* for ECR and *Edit Distance* for SR. The next section reports on an experiment to evaluate these two feature spaces.

5 Evaluation

A series of simulated dialogues was used to evaluate the two resulting feature spaces via the policies derived using them. These simulations were based on five-fold cross-validation, as in prior work (Henderson et al. 2008), with policies trained on four of the five folds and simulated users learned from the remaining fold.

As noted above, the rewards in the MDP were based on student learning gain, but learning gain (like user satisfaction in other dialogue domains) is not directly observable during the dialogues. However, we found that students in the high learning gain group had fewer *non-zero* task actions (actions that changed the edit distance to the final task solution) than students in the low learning gain group ($p < 0.05$). Therefore, number of non-zero task actions is used as a measure of dialogue success, with lower numbers being better. We derived the average change in edit distance on each state transition from the testing folds, and defined that a simulated dialogue would end when the edit distance reached zero (*i.e.*, the student arrived at the correct solution).

Table 3 shows the results of running 5,000 simulations in each fold for both the learned policy and for an anti-policy where each decision was reversed. The anti-policy is included to provide a point of comparison for the policies learned in each feature space, and offers insight into the quality of the learned policies, similar to the inverse policies learned in prior work (Chi et al. 2011). The table shows that the learned policies in the ECR feature space had slightly better results overall (lower number of non-zero task actions), while the SR feature space had larger separation between the learned policies and anti-policies. These results suggest that feature selection based on SR was able to identify important decision criteria with only a minor decrease in reward compared to ECR.

Feature Space	**Policy**	**Average non-zero task action count**
ECR	Learned policy	43.2
	Anti-policy	49.6
SR	Learned policy	47.3
	Anti-policy	97.4

Table 3. Results of simulated dialogues (lower non-zero task action count is better)

6 Conclusion

Modeling unrestricted turn taking within an RL framework, particularly for task-oriented dialogue with both a dialogue and a parallel task stream, presents numerous challenges. This paper has presented a novel representation of such dialogue with a tutoring domain, and has presented and evaluated a feature selection method based on a new *Separation Ratio* metric, which can inform the development of turn-taking policies in dialogue systems. Future work includes a more fine-grained analysis of the timing of dialogue moves as well as an evaluation of these results in a deployed system.

Acknowledgements

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

Chi, M., VanLehn, K., Litman, D., and Jordan, P. (2011). An Evaluation of Pedagogical Tutorial Tactics for a Natural Language Tutoring System: a Reinforcement Learning Approach. *International Journal of Artificial Intelligence in Education*, 21(1), 83–113.

D'Mello, S.K., Olney, A., and Person, N. (2010). Mining Collaborative Patterns in Tutorial Dialogues. *Journal of Educational Data Mining*, 2(1), 1–37.

English, M.S. and Heeman, P.A. (2005). Learning Mixed Initiative Dialog Strategies By Using Reinforcement Learning On Both Conversants. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 1011–1018.

Forbes-Riley, K. and Litman, D.J. (2012). Adapting to Multiple Affective States in Spoken Dialogue. In *Proceedings of the 13th Annual SIGDIAL Meeting on Discourse and Dialogue*, 217–226.

Henderson, J., Lemon, O., and Georgila, K. (2008). Hybrid Reinforcement/Supervised Learning of Dialogue Policies from Fixed Data Sets. *Computational Linguistics*, 34(4), 487–511.

Li, L., Williams, J. D., and Balakrishnan, S. (2009). Reinforcement Learning for Dialog Management Using Least-Squares Policy Iteration and Fast Feature Selection. In *Proceedings of the Conference of the International Speech Communication* Association. 2475–2478.

Misu, T., and Kashioka, H. (2012). Simultaneous Feature Selection and Parameter Optimization for Training of Dialog Policy by Reinforcement Learning. In *Proceedings of the IEEE Workshop on Spoken Language Technology*, 1–6.

Raux, A. and Eskenazi, M. (2012). Optimizing the Turn-Taking Behavior of Task-Oriented Spoken Dialog Systems. *Transactions on Speech and Language Processing*, 9(1), 1–23.

Selfridge, E.O., Arizmendi, I., Heeman, P.A., and Williams, J.D. (2012). Integrating Incremental Speech Recognition and POMDP-based Dialogue Systems. In *Proceedings of the 13th Annual SIGDIAL Meeting on Discourse and Dialogue*, 275–279.

Shute, V.J. (2008). Focus on Formative Feedback. *Review of Educational Research*, 78(1), 153–189.

Singh, S., Litman, D., Kearns, M., and Walker, M. (2002). Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. *Journal of Artificial Intelligence Research*, 16, 105–133.

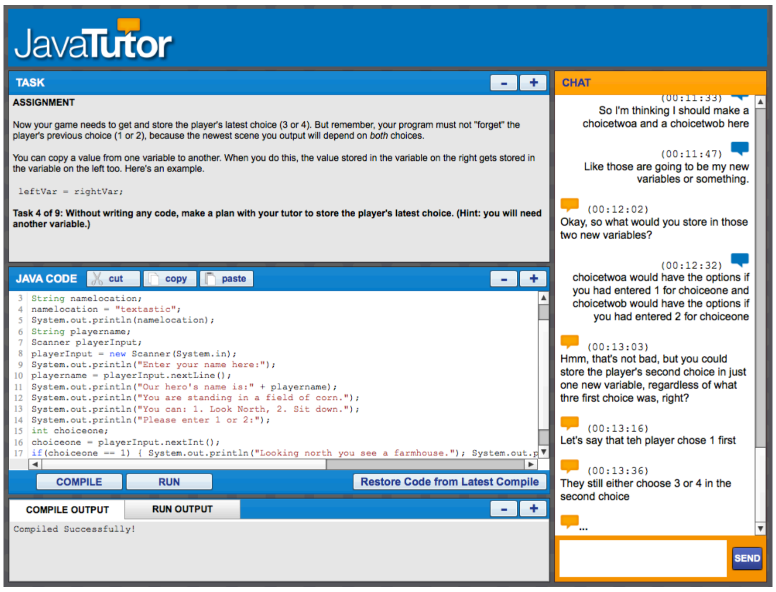
Sutton, R. and Barto, A. (1998). *Reinforcement Learning*. MIT Press, Cambridge, MA, 1998.

Tetreault, J.R. and Litman, D.J. (2008). A Reinforcement Learning Approach to Evaluating State Representations in Spoken Dialogue Systems. *Speech Communication*, 50(8), 683–696.

Williams, J.D. and Young, S. (2007). Partially Observable Markov Decision Processes for Spoken Dialog Systems. *Computer Speech & Language*, 21(2), 393–422.

Appendix A. Corpus excerpt

1. *Student begins declaring a String variable.*
2. *Student starts typing a message.*
3. **Student message:** Could I type in String The Adventure Quest; ? or would I need to put in quotes or something?
4. *Student resumes working on task.*
5. *Tutor starts typing a message.*
6. **Tutor message:** TheAdventureQuest is fine
7. *Student declares variable called* `The Adventure Quest` *(Incorrect Java syntax)*
8. *Tutor starts typing a message.*
9. *Student catches mistake and renames variable to* `TheAdventureQuest`
10. **Tutor message:** *Can't have spaces :)*
11. *Tutor starts typing a message*
12. **Tutor message:** Good job



ASSIGNMENT

Now your game needs to get and store the player's latest choice (3 or 4). But remember, your program must not "forget" the player's previous choice (1 or 2), because the newest scene you output will depend on *both* choices.

You can copy a value from one variable to another. When you do this, the value stored in the variable on the right gets stored in the variable on the left too. Here's an example.

```
leftVar = rightVar;
```

Task 4 of 9: Without writing any code, make a plan with your tutor to store the player's latest choice. (Hint: you will need another variable.)

```
String namelocation;
namelocation = "textastic";
System.out.println(namelocation);
String playername;
Scanner playerInput;
playerInput = new Scanner(System.in);
System.out.println("Enter your name here:");
playername = playerInput.nextLine();
System.out.println("Our hero's name is:" + playername);
System.out.println("You are standing in a field of corn.");
System.out.println("You can: 1. Look North, 2. Sit down.");
System.out.println("Please enter 1 or 2:");
int choiceone;
choiceone = playerInput.nextInt();
if(choiceone == 1) { System.out.println("Looking north you see a farmhouse."); System.out.p
```

```
Compiled Successfully!
```

(00:11:33)
So I'm thinking I should make a choicetwoa and a choicetwob here

(00:11:47)
Like those are going to be my new variables or something.

(00:12:02)
Okay, so what would you store in those two new variables?

(00:12:32)
choicetwoa would have the options if you had entered 1 for choiceone and choicetwob would have the options if you had entered 2 for choiceone

(00:13:03)
Hmm, that's not bad, but you could store the player's second choice in just one new variable, regardless of what thre first choice was, right?

(00:13:16)
Let's say that teh player chose 1 first

(00:13:36)
They still either choose 3 or 4 in the second choice

...

Appendix B. Dialogue interface