

Representing the Reinforcement Learning State in a Negotiation Dialogue

Peter A. Heeman

Center for Spoken Language Understanding
Oregon Health & Science University
Beaverton OR, 97006, USA
heemanp@ohsu.edu

Abstract—Most applications of Reinforcement Learning (RL) for dialogue have focused on slot-filling tasks. In this paper, we explore a task that requires negotiation, in which conversants need to exchange information in order to decide on a good solution. We investigate what information should be included in the system’s RL state so that an optimal policy can be learned and so that the state space stays reasonable in size. We propose keeping track of the decisions that the system has made, and using them to constrain the system’s future behavior in the dialogue. In this way, we can compositionally represent the strategy that the system is employing. We show that this approach is able to learn a good policy for the task. This work is a first step to a more general exploration of applying RL to negotiation dialogues.

I. INTRODUCTION

Reinforcement Learning (RL) is becoming a popular tool for building dialogue managers. In this approach, one specifies all of the different states that the dialogue manager can be in, all of the different actions it can perform, and a cost function that specifies how good a dialogue is. One also gives it a model of how the user behaves. RL then determines a *dialogue policy*: what action is best to perform in each dialogue state.

RL has mainly been used for system-initiative form-filling dialogues (i.e., [1], [2], [3]). In these dialogues, the system asks the user values for a number of slots. There might also be speech recognition errors. The system needs to decide which parameters to ask, what order to ask them in, and which ones (if any) to confirm. For such tasks, it is easy to specify the state of the system, as it can simply capture whether each slot has been asked, and whether its value has been confirmed.

Our long term goal is to expand the use of RL to more complex tasks, such as negotiation dialogues, in which the system and the user need to exchange information in order to decide on a good solution. The information though, unlike in a form-filling dialogue, is not part of the final solution. Given the indirect role that information exchange plays, how do we represent it as part of the state of the system? We need to capture enough details about how information was exchanged and how the solution was arrived at so that we can reward strategies that result in good solutions and penalize strategies that result in bad solutions. We also need to ensure that the number of state-action pairs does not explode.

Setting up an actual system to perform a negotiation task with real users is extremely time consuming. Furthermore, it introduces a number of tangential issues, such as speech recognition performance, parsing, and semantic interpretation.

Hence, as a first step, we use an artificial negotiation task and artificial users. For this task, we find that we are able to learn a good (if not optimal) system policy for solving the negotiation task. We achieved this by having the RL state keep track of the process by which the solution was arrived. First, we keep a compact list of the key decisions that were made. Second, we restrict the system’s future actions to be consistent with these decisions. In our experiments, we show how well our solution does for keeping the number of state-action pairs manageable, and the detriment of using alternative state representations.

Although our results are for an artificial domain with artificial users, we feel that the manner in which the RL state tracks how negotiation process will apply to real negotiation tasks with actual users.

II. REINFORCEMENT LEARNING

RL learns a dialogue policy by determining what action is best for each possible state. It does this by estimating Q values for each state-action pair $s-a$, which is the cost of getting to the final state from state s initially using action a , and then following the current policy. RL starts with an initial policy, and runs large numbers of dialogue simulations in which it interleaves estimating the Q values with refining the policy.

The state space used by RL does not have to comprise all of the different knowledge states of the system. Most research in applying RL to dialogue distinguishes between the full knowledge of the system, and a smaller number of states that is used by RL (i.e., [2], [4], [5]). The full and the RL state spaces are often defined by a set of variables and the different combinations of values that the variables can have. Thus the RL state space is usually defined by a subset of the variables that define the system’s full knowledge [5].

In this paper, we focus on how the RL state space should be crafted. On the one hand, we need to ensure that the number of state-action pairs does not explode. In fact, we want to keep the number of RL states as small as possible. Fewer states means there will be fewer Q values to estimate, which should allow the optimal policy to be learned with less training.

On the other hand, the RL state space cannot be too small. First, RL needs to determine the best action for each state. As each state can only specify a single action, the state needs to be detailed enough so that the same action is appropriate for the entire state. In other words, the RL state needs to include any variables that might be used in deciding which

action to perform next. We refer to these as *action-decision variables*. We feel that dialogue designers can often identify these variables when setting up a dialogue problem for RL.

Second, in order to ensure that RL is able to find an optimal solution, the states must satisfy the two Markov Decision Process (MDP) assumptions.

$$\Pr(s_{i+1}|s_i a_i) = \Pr(s_{i+1}|s_1 a_1 \dots s_i a_i) \quad (1)$$

$$\Pr(c_i|s_i a_i) = \Pr(c_i|s_1 a_1 \dots s_i a_i) \quad (2)$$

The first assumption is that the probability of the next state depends only on the previous state and on what action was taken in that state, and does not depend on any of the states or actions prior to that. The second assumption is that the *cost* of transitioning to the next state only depends on the previous state and action, and not on any of the previous states or actions. Thus, if two sequences of actions, say $a_1 \dots a_i$ and $a'_1 \dots a'_j$, have different expected costs to get to the finish state, the two action sequences should result in two different states. We refer to any variable that is needed to meet the second MDP assumption, other than the action-decision variables, as a *bookkeeping variable*. It is these variables that are difficult to determine when setting up a dialogue problem in RL, especially negotiation dialogues.

III. RELATED WORK

A number of researchers have applied RL to learn the system behavior for a form-filling system-initiative task (i.e. [1], [2], [3]). For these tasks, the representation of the system's state has been very straight-forward. For each slot in the form, variables are used to track the system's knowledge about the slot, such as whether it knows the value of the slot, whether the user is adamant about its value, and whether the value has been confirmed or not. For a given value of the state variables, it does not matter how the system got to that point, as the cost to the end of the dialogue only depends on the system's knowledge of the slots. With such a state representation, good policies can be learned. Thus, the variables needed for the RL state are mainly action-decision variables, and few if any bookkeeping variables are needed.

Once we move away from slot-filling dialogues, characterizing the state becomes more difficult. Walker [6] used RL to learn the dialogue policy of a system that helps a user read email. Much of the system behavior is hard-coded. However, the system can choose between a number of ways of interacting with the user, such as whether it uses system-initiative (requesting specific pieces of information one at a time) or mixed-initiative (asking an open-ended question to the user). The choice for this option determines how the system behaves over a number of states. In all, there were 3 choices that the system made. Thus, in dealing with this more complex domain, a lot of the system behavior was hard-coded.

What is interesting about Walker's work is that she had RL make the choices for components of the dialogue strategy at the first utterance that it would have an effect. The choice is remembered as part of the RL state and constrains the system's future behavior. Our work will generalize this approach without having to hard-code so much of the system's behavior.

IV. FURNITURE LAYOUT TASK

We use the furniture layout task [7], [8], which is based on the DesignWorld task [9]. We use this task as it has been used in past studies and requires agents to share information and create a joint plan, which lends itself to rich dialogue behavior.

A. Task Description

The task requires two agents to agree on five furniture items to place in a room. Both agents know all the furniture items that can be chosen, which differ by color, type and value between 1 and 20. Each agent also has private preferences about which items it wants in the room; such as "if there is a red couch, there should also be a lamp." Each preference has a penalty between 1 and 20. The agents have the joint goal of finding five items that have the highest score, where the score is the sum of the values of the five items less the penalties for each violated preference of either agent.

The agents can do the following actions: PROPOSE, ACCEPT, REJECT, INFORM, and RELEASE TURN.¹ If no item has been proposed, either agent can PROPOSE an item, which makes that item the current proposal. If there is a current proposal, the agent who did not propose it can ACCEPT it. Either agent can REJECT it. Accepting an item results in that item being included in the solution and removes it as the current proposal. Rejecting an item removes it as the current proposal, but it still remains a valid choice for future proposals. In addition to accepting or rejecting a proposal, the non-proposer can INFORM the other of a preference that is violated by the current proposal. When an agent informs of a violated preference, that preference becomes mutually known. For turn-taking, we include the action RELEASE TURN, which the agent that currently has the turn can perform to signal that it is relinquishing the turn (cf. [11]). The inclusion of this action allows agents to perform multiple actions in a row, such as several informs followed by a reject.

The agents also do the following domain reasoning. First, to determine which item to propose, the agents use a greedy algorithm: it picks the item that, if added to the items accepted so far, results in the lowest score for the items. The score is computed as the sum of the items' values less any penalties from any violated preferences that the agent knows about. Second, to determine which violated preference to inform about, the agent first determines which item it would have proposed, and then determines which preferences are violated by the proposed item that are not violated by its choice. It then chooses the violated preference that has the highest penalty and is not known by the other agent.

For this paper, we added several additional restrictions to the furniture task. We purposely made the behavior of the user be simple so as to focus on learning the system behavior. The user always goes first. The user always proposes, while the system always informs, rejects and accepts. So as to avoid infinite loops, the system can only reject if it has made at

¹This set of actions is similar to that of [10] in her work on a language for describing collaboration.

least one inform, and can only release the turn after accepting or rejecting the proposed item.

Our setup of the furniture task is different from [7], as they employed learning on both the system and the user, and had the system either inform of all of the violated preferences or none of them. Furthermore, their dialogue policies did not always converge to the same average cost.

B. Encoding in the Information State Approach

Following earlier work [3], [5], we use the Information State approach to encode the full knowledge state of the system and the user and their dialogue processing. This approach uses a set of rules that update the information state to capture how an agent determines what to say next [12], [13].

The information state of each agent includes: SOLUTION to track the items that have been agreed upon, MYPREFS to track the agent’s private preferences, SHAREDPREFS to track the shared preferences, PROPOSED tracks which item, if any, has been proposed, PROPOSEDBY tracks who proposed it, and INFORMED tracks whether any violated preferences have been communicated for the current proposal.

The agent’s domain reasoning updates the following variables. If there is no proposed item, the variable BEST indicates which item the agent thinks is best to add. If there is an item proposed by the other agent, the variable BETTER indicates which item, if any, when added to the current solution results in a better solution than the proposed item, according to the preferences that the agent knows about. The variable WVPREF indicates which preference that the non-proposer thinks is the worst violated preference.

There are 6 action rules, PROPOSE, ACCEPT, REJECT, INFORM, RELEASE TURN, and FINISH. These rules have preconditions on them to enforce the behavior discussed in the previous section. For example, an item can only be proposed if there is not already an item proposed, and an item can only be rejected if one of the agents has informed the other of a violated preference. Note that these preconditions are not sufficient. For any dialogue state, several action rules can usually apply, which is what RL will be used to resolve [5].

We also have an understanding rule for each of the actions. The understanding rules update the information state variables based on what was last said. The understanding rule for PROPOSE updates the PROPOSED and PROPOSEDBY variables. The rule for REJECT resets the variables PROPOSED, PROPOSEDBY, and INFORMED. The rule for ACCEPT also resets those same variables, as well as updates SOLUTION. The rule for INFORM updates INFORMED and SHAREDPREFS, and for the agent that uttered the inform also updates MYPREFS.

We also use several deliberation rules. These are invoked after the understanding rules. These rules compute the values of BEST, BETTER, and WVPREF.

C. Encoding in RL

For implementing the problem in RL, we use a subset of the IS variables as our RL state. We include INFORMED and PROPOSEDBY. We include boolean versions of PROPOSED

and BETTER: PROPOSEDP indicates if anything has been proposed, and BETTERP indicates if a better item has been found. We also include a variant of WVPREF: WVPREFQ indicates the penalty of the worst violated preference quantized into 4 groups: 0, 1-3, 4-8, 9-20. We also include DONE, which indicates whether the agents have completed the task (by accepting the required number of items). These variables are all potential action-decision variables.

For scoring a dialogue, we use a weighted average of dialogue length (with weight 0.15) and solution quality (weight 1.0) [7]. We actually normalize the solution quality. We use a greedy algorithm to determine the ‘optimal’ solution that would be proposed if all of the preferences were mutually known. We then set solution quality as the difference between the score for the proposed solution and the ‘optimal’ solution.²

D. Hand-Crafted Policies

So that we will have a comparison for the policies that RL learns, we hand-crafted a number of dialogue policies for the system. We then tested them on 1,000,000 different tasks. For each task, we created 15 furniture items, randomly assigning type, color, and value and we randomly created 25 preferences for each agent, involving the 15 furniture items.

Table I gives the results of three hand-crafted policies. For the first one, labeled ‘Uncollaborative’, the system always accepts what the user proposed. We see that these dialogues are relatively short, but have a high solution quality cost, leading to an overall cost of 30.68. For the second one, ‘Random’, the system randomly chooses an applicable action. When the user suggests an item, the system will randomly decide whether to inform of a violated preference, accept the item, or reject it, depending on which of these are applicable.

TABLE I
BASELINE RESULTS

	Dialogue Length	Solution Quality	Dialogue Cost
Uncollaborative	3.30	27.38	30.68
Random	8.72	19.61	28.58
Inform if ≥ 9	7.85	3.95	11.80

We experimented with a large number of alternatives in order to try to determine which one was best. We found that a good strategy is one where the system informs of all preferences that have a penalty of at least 9, and not to inform of any other preferences. If it has informed of any preferences, it should reject, otherwise it should accept. We found that this achieved an overall cost of 11.80. Of course, for more complicated domains, experimenting with hand-crafted policies will not work, as there will be too many options to explore, which is precisely why RL is needed.

V. MODELING STRATEGY

In Section II, we described three criteria for determining if the RL state space includes enough distinctions. The problem with the RL variables that we proposed for the furniture task

²As we are using a greedy algorithm, the ‘optimal’ solution might not actually be optimal. Hence the proposed solution might be better than the ‘optimal’. In this case, we set the solution quality cost to 0.

is with the third criteria (Eqn. 2). The RL variables do not capture the process by which the system agrees to an item. For example, consider the following strategies by which the system might agree to a solution: (a) just accept each item that is proposed; (b) if there are any uncommunicated violated preferences for the current proposal, inform the user and reject the proposal. Which of these two strategies is used will result in different average solution qualities, and so will affect the cost from any intermediate state to the final state. Thus, we are missing some bookkeeping variables.

How do we encode the process by which items are agreed upon? One way to meet Eqn. 2 is to encode all of the past states and the chosen actions into the RL state. However, this would explode the RL state space of the system. Another option is to try to enumerate each possible strategy for completing the task, and at the beginning of a dialogue, have an action that sets an RL variable to one of the enumerated strategies, and future actions would be chosen in accordance with the selected strategy. However, as we apply RL to more complex domains, it might not be feasible to quantify all of the different strategies. This approach of setting the strategy at the beginning of the dialogue also suffers as each strategy will use a completely separate subset of states, thus requiring RL to estimate the strengths of each strategy independently of the others, negating one of the chief advantages of RL.

a) Recording the Past: The approach that we take in this paper is similar to that of Walker [6]. Rather than enumerate all possible strategies in advance, we enumerate components of a strategy in terms of what action is chosen in a certain situation. Each time the system makes a choice, it adds a condensed version of what the state is along with what action it picked. We store this in the variable STRATEGY that is part of the RL state. Thus we keep track of what the system has done in various situations.

We record decisions in the STRATEGY variable as a set of predicates, where the predicate name is the action chosen and the arguments are the conditions in which it was chosen. The conditions are represented by a subset of the RL state variables. For the furniture domain, we use WVPREF and INFORMED. Consider the case where the user proposes an item and the system has a worst uncommunicated violated preference of 1, and the system accepts it. The STRATEGY variable will record this as ACCEPT(1,0). If the next item that the user proposes has a worst violated preference of 4, and the system informs of it, STRATEGY is now ACCEPT(1,0) INFORM(4,0). If the system now has a worst violated preference of 1, and the system rejects it, STRATEGY is now ACCEPT(1,0) REJECT(1,1) INFORM(4,0). Note that the order that the actions were performed is not recorded, as the STRATEGY variable is an unordered set. (To emphasize that point, we show the value as an alphabetically ordered list.) Our use of the STRATEGY variable allows RL to reward action sequences that result in a good solution and penalize those that result in a bad solution.

The STRATEGY variable is keeping track of what actions we have taken. However, it is not keeping a full history and

so it is not tracking how many items we have agreed to so far. The number of items that are still remaining to decide is an important factor of what the cost is to the final state. Hence, we also include in our RL state the number of items agreed to so far, NUMAGREED.

b) Constraining Future Actions: The STRATEGY variable not just records a history of what the system has done so far in the dialogue, it also constrains what the system will do in the future in that dialogue. If the system sees a situation that it already encountered earlier in the dialogue, it will do the action that it previously did. For example, if STRATEGY is ACCEPT(1,0) REJECT(1,1) INFORM(4,0), and the system is now presented with a proposed item from the user with a worst violated preference of 4, the system must inform the user of the violated preference.

By constraining the future actions, the STRATEGY variable is restricting the range of policies that RL considers for the system during training.³ This is because of the following. First, a lot of the states will only have a single action that is applicable, thus reducing the number of state-actions pairs that are needed. Second, as we are restricting what actions the system can take, the number of different decisions that we need to track is also much less, which means less values for STRATEGY, and so fewer RL states.

Our approach of constraining the future actions forces the system to behave in a consistent manner: the process by which it accepts one item should be the same as for the others. One potential drawback is that the optimal policy for the system might require it to behave differently for agreeing to one item than it does for another item. Our approach will preclude it from finding such a solution.

c) Inferences: The third part of our approach is that we use inference in constructing the value of STRATEGY. If the system makes a certain action in one type of situation, we might want to constrain how it behaves in a related situation. Consider the case where the user proposes an item, the system has an uncommunicated violated preference against the item of penalty 4, and the system informs the user of that preference. If it makes sense to inform of a penalty of 4, it should also make sense to inform when the worst violated preference is 9. Likewise, if it makes sense to accept when there is a worst violated preference of 4, then it also makes sense to accept if the worse violated preference is 1. For the furniture task, we use 4 rules for deducing additional constraints for the STRATEGY variable.

- If INFORM(W,I) is in STRATEGY, add INFORM(X,I) for all $x > w$
- If ACCEPT(W,I) is in STRATEGY, add ACCEPT(X,I) for all $x < w$
- If REJECT(W,I) and REJECT(Y,I) are in STRATEGY, add REJECT(X,I) for all x s.t. $w \leq x < Y$
- If INFORM(W,1) is in STRATEGY, add INFORM(W,0)

³During testing, this restriction is redundant, as the system will follow the action that is specified in the policy.

The inferences complicate how we determine whether a system action is applicable. This is because choosing a particular action might allow us to infer a contraction in the values of the STRATEGY variable, where two different actions are allowed for the same situation. Hence, we incorporate sophisticated checks to ensure that this does not happen in the preconditions of the system actions. This checking does not add much expense to learning a dialogue policy because after RL encounters an RL state for the first time, it caches what actions are applicable for that state, and hence does not need to subsequently check the action’s preconditions.

As we show in Section VI-B, the use of inferences reduces the state and state-action spaces. The inferences are knowledge that the system should be able to discover with RL; however, including them allows us to keep the search space smaller. Furthermore, the use of inferences highlights that the STRATEGY variable does not just have to be a record of what occurred in the dialogue, but can capture the strategy in a compositional way from individual system actions.

VI. EXPERIMENTS

In this section, we present the results from a set of experiments that show the importance of how the state is constructed. We start with our proposed RL state, and then show the effect of removing variables (or distinctions) from it. In all experiments, we use Q-learning with ϵ -greedy to explore the state-action pair space, with ϵ at 20%. New training experiences are given a weight of $\frac{1}{\sqrt{c}}$, where c is the number of times the state-action pair has been visited. We group training into epochs of 100 dialogue runs; after each epoch, a new policy is determined. After every 1000 epochs (and after 10, 100, and 500 epochs), we test the policy with 5,000 dialogue runs. For testing, we always follow the action in the policy, with no exploration.

A. Strategy Variable With Inferences

We first use the RL state representation of Section V, which includes the STRATEGY and NUMAGREED variables. We trained 12 policies for each of 5 different problem sizes, from agreeing to a single furniture item to agreeing to five items. Each policy was trained for 200,000 epochs. The first issue we investigate is how the size of the RL state scales up as the problem size increases. As our RL state includes NUMAGREED (which tracks how many items have been agreed to so far), the number of states and state-action pairs increases linearly with the size of the problem. In Figure 1, we show the average number of states and state-action pairs visited after 200,000 epochs of training for the five problem sizes.⁴

The second issue we investigate is whether this state representation allows RL to find an optimal policy. We found that we were able to learn a policy that performed as well as our best hand-crafted strategies (Section IV-D) for each of the

⁴For both the number of states and state-action pairs, the initial part of the curve is not linear. This is because some values of STRATEGY do not occur when the number of items that has been agreed to so far is 0 or 1 when the number of items discussed (NUMAGREED) is one or two items.

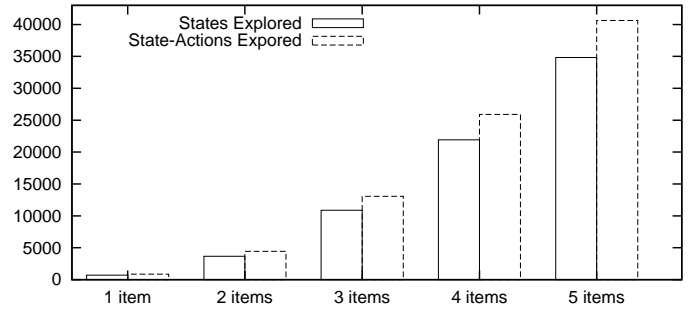


Fig. 1. State-action space versus problem size

five problem sizes. Even though the RL state space is only growing linearly with the solution size, the state space is able to capture the process by which the solution is agreed upon.

B. Removing Inferences

We next modified the STRATEGY variable to remove the inferences that the system makes. This means it must separately learn what to do for each preference value. Thus, more states and state-actions must be estimated, and so more policies are searched over. We trained 12 policies for 200,000 epochs. As shown in the second and third columns of Table II, more than twice as many states and state-action pairs are needed.

TABLE II
COMPARISON OF PROPOSED VERSION WITH ALTERNATIVES

	Proposed State	Without Inferences	Without NumAgreed	Without Strategy
States	34,838	81,883	14,087	51
State-Actions	40,629	99,946	16,086	101
Dialogue Cost	11.825	11.822	12.067	30.716

Even though there are more states and state-actions pairs, RL should still be able to find a policy that performs just as well as the version that uses inferences. However, we should expect RL to take longer to converge on an optimal policy. In Figure 2, we show the average dialogue cost for the two sets of learned policies as training progresses. Here, we can clearly see that the version without the inferences takes longer to converge.

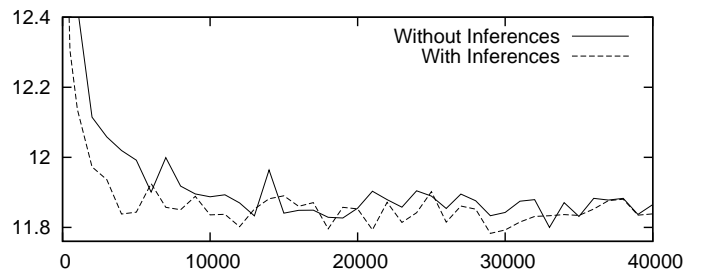


Fig. 2. Effect of removing the strategy inferences

C. Removing the NumAgreed Variable

We next ran a version in which we included the STRATEGY variable with the inferences (similar to Section VI-A), but excluded NUMAGREED, which tracks how many items the conversants have agreed on so far. This allows us to determine whether NUMAGREED is playing a useful role. We trained

12 policies for 60,000 epochs. As can be seen by comparing the second and fourth columns of Table II, excluding NUMAGREED results in a substantial reduction in the number of states and state-action pairs. In fact, the number of states and state-action pairs no longer increases as the problem size increases.

As there are substantially less states and state-action pairs, there are far fewer Q values to estimate. This usually allows RL to more quickly learn a good policy. However, as can be seen in Figure 3, RL is not able to hone in on an optimal policy, only achieving an average cost of 12.067 rather than 11.825. In fact, RL seems to stop learning after 3,000 epochs.

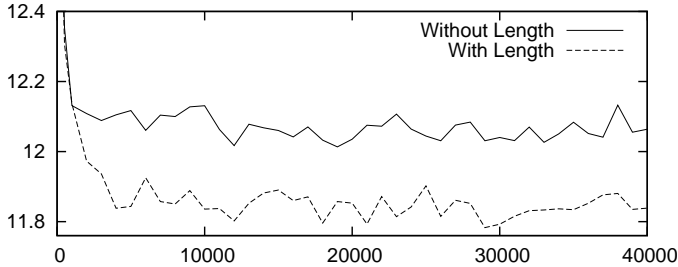


Fig. 3. Removing the NumAgreed variable

The above experiment shows that we need to track the progress of the system in terms of how many items have been agreed to. The inclusion of NUMAGREED allows RL to better estimate the number of utterances to finish the problem. Thus, the size of the state and state-action space do need to increase as the task becomes larger.

D. Removing the Strategy Variable

We next removed the STRATEGY variable from the RL state, but included NUMAGREED. This allows us to determine whether the strategy variable is playing a useful role. We trained 12 policies for 20,000 epochs. As can be seen by comparing the second and fifth columns of Table II, removing the strategy variable reduces the number of states and state-action pairs dramatically. Thus, there are far fewer probability distributions to estimate, which usually leads to faster learning.

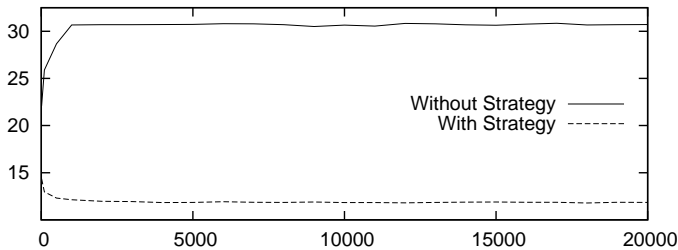


Fig. 4. Removing the Strategy variable

As can be seen in Figure 4, the reduction in the number of states and state-action pairs does not lead to faster convergence. Instead, the cost of the trained policies initially gets worse with more training until stabilizing around 1000 epochs with a cost of around 30.6, in which it settles on the uncollaborative policy of Section IV-D. RL does not seem to be able to learn a reasonable policy at all. The initial

increase in cost is probably because we start RL with a policy learned from a random exploration of actions for each state. With subsequent learning, it actually finds an even worse policy. Without the STRATEGY variable, RL does not take into account the process by which the conversants agree to items in the plan.

VII. CONCLUSION

In this paper, we showed how to construct the RL state for a negotiation task. We included bookkeeping variables to track in a concise way the process by which the system negotiates with the user. We added the variable STRATEGY to track of the key decisions that were made, and use this to constrain the system future actions, so that the the system negotiates in a consistent way. This consistency also lets us limit the number of the state-action pairs. The key decisions can even include inferences to further limit number of the state-action pairs. We also included NUMAGREED to track how far we are in the negotiation. Although this work was on an artificial negotiation task, we feel that the manner in which the RL state tracks how negotiation process will apply to real negotiation tasks with actual users.

VIII. ACKNOWLEDGMENT

The author gratefully acknowledges funding from the National Science Foundation under grant IIS-0713698 and IIS-0931338, and helpful discussions with Rebecca Lunsford, Ethan Selfridge, Andrew Rueckert, and Jordan Fryer.

REFERENCES

- [1] E. Levin, R. Pieraccini, and W. Eckert, "A stochastic model of human-machine interaction for learning dialog strategies," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 1, pp. 11–23, 2000.
- [2] K. Scheffler and S. J. Young, "Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning," in *Proceedings of Human Language Technology*, San Diego CA, 2002, pp. 12–18.
- [3] K. Georgila, J. Henderson, and O. Lemon, "Learning user simulations for information state update dialogue systems," in *Proceedings of InterSpeech*, Lisbon Portugal, Sep. 2005.
- [4] S. Singh, D. Litman, M. Kearns, and M. Walker, "Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system," *J. of Artificial Intelligence Research*, vol. 16, pp. 105–33, 2002.
- [5] P. Heeman, "Combining reinforcement learning with information-state update rules," in *Proc. of NAACL*, Rochester NY, 2007, pp. 268–75.
- [6] M. Walker, "An application of information state learning to dialogue strategy selection in a spoken dialogue system for email," *Journal of Artificial Intelligence Research*, vol. 12, pp. 387–416, 2000.
- [7] M. English and P. Heeman, "Learning mixed initiative dialog strategies by using reinforcement learning on both conversants," in *Proceedings of the HLT-EMNLP*, Vancouver Canada, Oct. 2005, pp. 1011–18.
- [8] F. Yang and P. Heeman, "Using computer simulation to compare two models of mixed-initiative," in *Proc. of InterSpeech*, 2004, pp. 213–16.
- [9] M. Walker, "Testing collaborative strategies by computational simulation: Cognitive and task effects," *Knowledge-Based Systems*, vol. 8, pp. 105–16, 1995.
- [10] C. Sidner, "An artificial discourse language for collaborative negotiation," in *Proceedings of AAAI*, 1994, pp. 814–19.
- [11] D. Traum and E. Hinkelman, "Conversation acts in task-oriented spoken dialogue," *Computational Intelligence*, vol. 8, no. 3, pp. 575–99, 1992.
- [12] P. Bohlin, R. Cooper, E. Engdahl, and S. Larsson, "Information states and dialog move engines," *Electronic Transactions in AI*, vol. 3, no. 9, pp. 53–71, 1999.
- [13] S. Larsson and D. Traum, "Information state and dialogue management in the TRINDI dialogue move engine toolkit," *Natural Language Engineering*, vol. 6, pp. 323–40, 2000.