Reinforcement Learning in Multi-Party Trading Dialog

Takuya Hiraoka
Nara Institute of Science and Technology
takuya-h@is.naist.jp

Kallirroi Georgila
USC Institute for Creative Technologies
kgeorgila@ict.usc.edu

Elnaz Nouri
USC Institute for Creative Technologies
nouri@ict.usc.edu

David Traum
USC Institute for Creative Technologies
traum@ict.usc.edu

Satoshi Nakamura
Nara Institute of Science and Technology
s-nakamura@is.naist.jp

Abstract

In this paper, we apply reinforcement learning (RL) to a multi-party trading scenario where the dialog system (learner) trades with one, two, or three other agents. We experiment with different RL algorithms and reward functions. The negotiation strategy of the learner is learned through simulated dialog with trader simulators. In our experiments, we evaluate how the performance of the learner varies depending on the RL algorithm used and the number of traders. Our results show that (1) even in simple multi-party trading dialog tasks, learning an effective negotiation policy is a very hard problem; and (2) the use of neural fitted Q iteration combined with an incremental reward function produces negotiation policies as effective or even better than the policies of two strong hand-crafted baselines.

1 Introduction

Trading dialogs are a kind of interaction in which an exchange of ownership of items is discussed, possibly resulting in an actual exchange. These kinds of dialogs are pervasive in many situations, such as marketplaces, business deals, school lunchrooms, and some kinds of games, like Monopoly or Settlers of Catan (Guhe and Lascarides, 2012). Most of these dialogs are non-cooperative (Traum, 2008; Asher and Lascarides, 2013), in the sense that mere recognition of the desire for one party to engage in a trade does not provide sufficient inducement for the other party to accept the trade. Usually a trade will only be accepted if it is in the perceived interest of each party. Trading dialogs can be considered as a kind of negotiation, in which participants use various tactics to try to reach an agreement. It is common to have dialogs that may involve multiple offers or even multiple trades. In this way, trading dialogs are different from other sorts of negotiation in which a single decision (possibly about multiple issues) is considered, for example partitioning a set of items (Nouri et al., 2013; Georgila et al., 2014). Another difference between trading dialogs and partitioning dialogs is what happens when a deal is not made. In partitioning dialogs, if an agreement is not reached, then participants get nothing, so there is a very strong incentive to reach a deal, which allows pressure and can result in a "chicken game", where people give up value in order to avoid a total loss. By contrast, in trading dialogs, if no deal is made, participants stick with the status quo. Competitive two-party trading dialogs may result in a kind of stasis, where the wealthier party will pass up mutually beneficial deals, in order to maintain primacy. On the other hand, multi-party trading dialogs involving more than two participants changes the dynamic again, because now a single participant cannot necessarily even block another from acquiring a missing resource, because it might be available through trades with a third party. A player who does not engage in deals may lose relative position, if the other participants make mutually beneficial deals.

In this paper, we present a first approach toward learning dialog policies for multi-party trading dialogs. We introduce a simple, but flexible game-like scenario, where items can have different values for different participants, and also where the value of an item can depend on the context of other items held. We examine a number of strategies for this game, including random, simple, and complex

Proceedings of the SIGDIAL 2015 Conference, pages 32–41,
Prague, Czech Republic, 2-4 September 2015.

hand-crafted strategies, as well as several reinforcement learning (RL) (Sutton and Barto, 1998) algorithms, and examine performance with different numbers and kinds of opponents.

In most of the previous work on statistical dialog management, RL was applied to cooperative slot-filling dialog domains. For example, RL was used to learn the policies of dialog systems for food ordering (Williams and Young, 2007a), tourist information (Williams and Young, 2007b), flight information (Levin et al., 2000), appointment scheduling (Georgila et al., 2010), and e-mail access (Walker, 2000). In these typical slot-filling dialog systems, the reward function depends on whether the user's goal has been accomplished or not. For example, in the food ordering system presented by Williams and Young (2007a), the dialog system earns higher rewards when it succeeds in taking the order from the user.

Recently, there has been an increasing amount of research on applying RL to negotiation dialog domains, which are generally more complex than slot-filling dialog because the system needs to consider its own goal as well as the user's goal, and may need to keep track of more information, e.g., what has been accepted or rejected so far, proposals and arguments on the table, etc. Georgila and Traum (2011) applied RL to the problem of learning negotiation dialog system policies for different cultural norms (individualists, collectivists, and altruists). The domain was negotiation between a florist and a grocer who had to agree on the temperature of a shared retail space. Georgila (2013) used RL to learn the dialog system policy in a two-issue negotiation domain where two participants (the user and the system) organize a party, and need to decide on both the day that the party will take place and the type of food that will be served. Also, Heeman (2009) modeled negotiation dialog for a furniture layout task, and Paruchuri et al. (2009) modeled negotiation dialog between a seller and buyer. More recently, Efstathiou and Lemon (2014) focused on non-cooperative aspects of trading dialog, and Georgila et al. (2014) used multi-agent RL to learn negotiation policies in a resource allocation scenario. Finally, Hiraoka et al. (2014) applied RL to the problem of learning cooperative persuasive policies using framing, and Nouri et al. (2012) learned models for cultural decision-making in a simple negotiation game (the Ultimatum Game). In contrast to typical slot-filling dialog systems, in these negotiation dialogs, the dialog system is rewarded based on the achievement of its own goals rather than those of its interlocutor. For example, in Georgila (2013), the dialog system gets a higher reward when its party plan is accepted by the other participant.

Note that in all of the previous work mentioned above, the focus was on negotiation dialog between two participants only, ignoring cases where negotiation takes place between more than two interlocutors. However, in the real world, multi-party negotiation is quite common. In this paper, as a first study on multi-party negotiation, we apply RL to a multi-party trading scenario where the dialog system (learner) trades with one, two, or three other agents. We experiment with different RL algorithms and reward functions. The negotiation strategy of the learner is learned through simulated dialog with trader simulators. In our experiments, we evaluate how the performance of the learner varies depending on the RL algorithm used and the number of traders. To the best of our knowledge this is the first study that applies RL to multi-party (more than two participants) negotiation dialog management. We are not aware of any previous research on dialog using RL to learn the system's policy in multi-party negotiation.[1]

Our paper is structured as follows. Section 2 provides an introduction to RL. Section 3 describes our multi-party trading domain. Section 4 describes the dialog state and set of actions for both the learner and the trader simulators, as well as the reward functions of the learner and the hand-crafted policies of the trader simulators. In Section 5, we present our evaluation methodology and results. Finally, Section 6 summarizes the paper and proposes future work.

2 Reinforcement Learning

Reinforcement learning (RL) is a machine learning technique for learning the policy of an agent

[1]Note that there is some previous work on using RL to learn negotiation policies among more than two participants. For example, Mayya et al. (2011) and Zou et al. (2014) used multi-agent RL to learn the negotiation policies of sellers and buyers in a marketplace. Moreover, Pfeiffer (2004) used RL to learn policies for board games where sometimes negotiation takes place among players. However, these works did not focus on negotiation dialog (i.e., exchange of dialog acts, such as offers and responses to offers), but only focused on specific problems of marketing or board games. For example, in Zou et al. (2014)'s work, RL was used to learn policies for setting selling/purchasing prices in order to achieve good payoffs.

that takes some action to maximize a reward (not only immediate but also long-term or delayed reward). In this section, we briefly describe RL in the context of dialog management. In dialog, the policy is a mapping function from a dialog state to a particular system action. In RL, the policy's goal is to maximize a reward function, which in traditional task-based dialog systems is user satisfaction or task completion (Walker et al., 1998). RL is applied to dialog modeling in the framework of Markov decision processes (MDPs) or partially observable Markov decision processes (POMDPs).

In this paper, we follow an MDP-based approach. An MDP is defined as a tuple $\langle S, A, P, R, \gamma \rangle$ where S is the set of states (representing different contexts) which the system may be in (the system's world), A is the set of actions of the system, $P : S \times A \rightarrow P(S, A)$ is the set of transition probabilities between states after taking an action, $R : S \times A \rightarrow \Re$ is the reward function, and $\gamma \in$ [0, 1] a discount factor weighting long-term rewards. At any given time step i the world is in some state $s_i \in S$. When the system performs an action $\alpha_i \in A$ following a policy $\pi : S \rightarrow A$, it receives a reward $r_i(s_i, \alpha_i) \in \Re$ and transitions to state s_{i+1} according to $P(s_{i+1}|s_i, \alpha_i) \in P$. The quality of the policy π followed by the agent is measured by the *expected future reward*, also called Q-function, $Q^\pi : S \times A \rightarrow \Re$.

We experiment with 3 different RL algorithms:

LinQ: This is the basic Q-learning algorithm with linear function approximation (Sutton and Barto, 1998). The Q-function is a weighted function of state-action features. It is updated whenever the system performs an action and gets a reward for that action (in contrast to batch RL mentioned below).

LSPI: In least-squares policy iteration (LSPI), the Q-function is also approximated by a linear function (similarly to LinQ). However, unlike LinQ, LSPI is a batch learning method. It samples the training data one or more times (batches) using a fixed system policy (the policy that has been learned so far), and the approximated Q-function is updated after each batch. We use LSPI because it has been shown to achieve higher performance than LinQ in some tasks (Lagoudakis and Parr, 2003).

NFQ: Neural fitted Q iteration (NFQ) uses a multi-layered perceptron as the Q-function approximator. Like LSPI, NFQ is a batch learning method. We introduce NFQ because it has been shown to perform well in some tasks (Riedmiller, 2005).

During training we use ϵ-greedy exploration, i.e., the system randomly selects an action with a probability of ϵ (we used a value of 0.1 for ϵ) otherwise it selects the action which maximizes the Q-function given the current state. During testing there is no exploration and the policy is dictated by the Q-values learned during training.

3 Multi-Party Trading Domain

Our domain is trading, where two or more traders have a number of items that they can keep or exchange with the other traders in order to achieve their goals. The value of each item for a trader is dictated by the trader's payoff matrix. So at the end of the interaction each trader earns a number of points based on the items that it holds and the value of each item. Note that each trader has its own payoff matrix. During the interaction, each trader can trade an item with the other traders (i.e., offer an item in exchange for another item). If the addressee of the offer accepts it, then the items of the traders involved in this exchange are updated. If the offer is not accepted, the dialog proceeds without any changes in the number of items that each trader possesses. To make the search space of possible optimal trading policies more tractable, we assume that each trader can only trade one item at a time, and also that each offer is addressed only to one other trader. Each trader can take the turn (decide to trade) in random order, unless there is a pending offer. That is, if a trader makes an offer to another trader, then the addressee of that offer has priority to take the next turn; the addressee can decide to accept the offer, or to do nothing, or to make a different offer. Note that the traders do not know each other's payoff matrices but they know the items that each trader owns. The dialog is completed after a fixed period of time passes or when all traders decide not to make any offers.

In our experiments, there are three types of items: apple, orange, and grape, and each trader may like, hate, or feel neutral about each type of fruit. At the end of the dialog the trader earns 100 points for each fruit that he likes, 0 points for each fruit that he is neutral to, and -100 points for each fruit that he hates. Payoff matrices are structured

such that there is always one fruit that each trader likes, one fruit that he is neutral to, and one fruit that he hates. Furthermore, all traders can get a big payoff for having a fruit salad, i.e., the trader earns 500 additional points if he ends up with one fruit of each type. Thus even hated fruits may sometimes be beneficial, but only if they can be part of a fruit salad. Thus the outcome for a trader o_{tr} is calculated by Equation (1).

$$\begin{aligned} o_{tr} &= Pay(\text{apple}_{\text{tr}}) * Num(\text{apple}_{\text{tr}}) \\ &+ Pay(\text{orange}_{\text{tr}}) * Num(\text{orange}_{\text{tr}}) \\ &+ Pay(\text{grape}_{\text{tr}}) * Num(\text{grape}_{\text{tr}}) \\ &+ Pay(\text{salad}_{\text{tr}}) \end{aligned} \quad (1)$$

$$Pay(\text{salad}_{\text{tr}}) = \begin{cases} 500 & \text{if } Num(\text{apple}_{\text{tr}}) \geq 1 \\ & \text{and } Num(\text{orange}_{\text{tr}}) \geq 1 \\ & \text{and } Num(\text{grape}_{\text{tr}}) \geq 1 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where Pay is a function which takes as argument a fruit type and returns the value of that fruit type for the trader, and Num shows the number of items of a particular fruit type that the trader possesses. At the beginning of each dialog, the initial conditions (i.e., number of items per fruit type and payoff matrix) of the traders (except for the learner) are randomly assigned. The learner always has the same payoff matrix for all dialogs, i.e., the learner always likes grape, always feels neutral about apple, and always hates orange. Also, the total number of fruits that the learner holds in the beginning of the dialog is always 3. However, the number of each fruit type that the learner holds is randomly initialized for each dialog, e.g., the learner could be initialized with (1 apple, 2 oranges, 0 grapes), or (1 apple, 1 orange, 1 grape), etc. The total number of fruits for each trader is determined based on his role (Rich: 4 items, Middle: 3 items, Poor: 2 items), which is also randomly assigned at the beginning of each dialog. Table 1 shows two example dialogs.

4 Methodology for Learning Multi-Party Negotiation Policies

In this section, we present our methodology for training the learner, including how we built our trader simulators. The trader simulators are used as negotiation partners of the learner for both training and evaluating the learner's policy (see Section 5).

4.1 Learner's Model

Below we define the reward function, sets of actions, and state of our MDP-based learner's model. Note that we use two kinds of rewards.

The first type of reward is based on Equation (3). In this case, the learner is rewarded based on its outcome only at the end of the dialog. In all other dialog turns i its reward is 0.

$$r_{end} = \begin{cases} o_{tr} & \text{if dialog ends} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

We also introduce an *incremental reward* for training, because rewarding a learning agent only at the end of the dialog makes the learning problem very difficult, thus sub-goals can be utilized to reward the learning agent incrementally (McGovern and Barto, 2001). The incremental reward at turn i is given by Equation (4), where $o_{tr}(i)$ is the outcome for a trader applied at time point i.

$$r_i^{'} = \begin{cases} \gamma * o_{tr}(i) - o_{tr}(i-1) & \text{if } i > 0 \\ 0 & \text{if } i = 0 \end{cases} \quad (4)$$

This equation represents the improvement on the outcome of the learner at turn i compared to its outcome at the previous turn $i-1$. Note that this implementation of the incremental reward function is basically the same as reward shaping, and has the following property (Ng et al., 1999; Asri et al., 2013): the policy learned by using Equation (4) maximizes the expectation of the cumulative reward given by Equation (3).

The learner's actions are presented below. By speaker we mean the trader who is performing the action. In this case, the speaker is the learner, but as we will see below this is also the set of actions that a trader simulator can perform.

Offer(A, I_s, I_a): offering addressee A to trade the speaker's item I_s for the addressee's item I_a.

Accept: accepting the most recent offer addressed to the speaker.

Keep: passing the turn without doing anything. If there is a pending offer addressed to the speaker, then this offer is rejected.

The dialog state consists of the *offered table* and the distribution of the items among the negotiators:

Offered table: The offered table consists of all possible tuples (Trading partner, Fruit requested, Fruit offered in return). If another

		Item			Outcome		
Speaker	Utterance	TR1	TR2	TR3	TR1	TR2	TR3
Dialog 1:							
1: TR1	TR2, could you give me an orange? I'll give you a grape. (Offer)	A: 0, O: 0, G: 3	A: 1, O: 1, G: 0	A: 0, O: 1, G: 2	0	-100	100
2: TR2	Okay. (Accept)	A: 0, O: 1, G: 2	A: 1, O: 0, G: 1	A: 0, O: 1, G: 2	100	0	100
Dialog 2:							
1: TR2	TR1, could you give me a grape? I'll give you a apple. (Offer)	A: 0, O: 0, G: 3	A: 1, O: 1, G: 0	A: 0, O: 1, G: 2	0	-100	100
2: TR1	I want to keep my fruits. (Keep)	A: 0, O: 0, G: 3	A: 1, O: 1, G: 0	A: 0, O: 1, G: 2	0	-100	100
3: TR3	TR2, could you give me an apple? I'll give you a grape. (Offer)	A: 0, O: 0, G: 3	A: 1, O: 1, G: 0	A: 0, O: 1, G: 2	0	-100	100
4: TR2	Okay. (Accept)	A: 0, O: 0, G: 3	A: 0, O: 1, G: 1	A: 1, O: 1, G: 1	0	100	500

Table 1: Examples of two trading dialogs among traders TR1, TR2, and TR3. In these examples, the payoff matrix of TR1 is (apple: -100, orange: 100, grape: 0), that of TR2 is (apple: -100, orange: 0, grape: 100), and that of TR3 is (apple: 0, orange: -100, grape: 100). Item and Outcome show the number of items per fruit type of each trader and the points that each trader has accumulated after an action. A stands for apple, O for orange, and G for grape.

agent makes an offer to the learner then the learner's offered table is updated. The dialog state is represented by binary variables (or features). In Example 1, we can see a dialog state in a 2-party dialog, after the learner receives an offer to give an orange and in return take an apple.

Number of items: The number of items for each fruit type that each negotiator possesses. Once a trade is performed, this part of the dialog state is updated in the dialog states of all agents involved in this trade.

4.2 Trader Simulator

In order to train the learner we need trader simulators to generate a variety of trading episodes, so that in the end the learner learns to follow actions that lead to high rewards and avoid actions that lead to penalties. The trader simulator has the same dialog state and actions as the learner. We have as many trader simulators as traders that the learner negotiates with. Thus in a 3-party negotiation we have 2 trader simulators. The policy of the trader simulator can be either hand-crafted, designed to maximize the reward function given by Equation (3); or random.

The hand-crafted policy is based on planning. More concretely, this policy selects an action based on the following steps:

1. Pre-compute all possible sets of items (called "hands", by analogy with card games, where

Example 1: Status of the learner's dialog state's features in a 2-party trading dialog (learner vs. Agent 1). Agent 1 has just offered the learner 1 apple for 1 of the learner's 2 oranges (but the learner has not accepted or rejected the offer yet). This is why the (Agent 1, orange, apple) tuple has value 1. Initially the learner has (O apples, 2 oranges, 1 grape) and Agent 1 has (1 apple, 0 oranges, 1 grape). Note that if we had more negotiators e.g., Agent 2, the dialog state would include features for offer tuples for Agent 2, and the number of items that Agent 2 possessed.

Trading partner	Item requested by partner	Item given by partner to learner	Occurrence binary value (used as feature)
Agent 1	apple	orange	0
	apple	grape	0
	orange	apple	1
	orange	grape	0
	grape	apple	0
	grape	orange	0

Agent who possesses fruits	Fruit type	Number of fruits (used as feature)
learner	apple	0
	orange	2
	grape	1
Agent 1	apple	1
	orange	0
	grape	1

each item is represented by a card), given the role of the trader (Rich, Middle, Poor) and how many items there can be in the hand.

2. Compute the valuation of each of the hands, according to the payoff matrix.
3. Based on the possible trades with the other agents, compute a set of achievable hands, and order them according to the valuations defined in step 2. A hand is "achievable" if there are enough of the right types of items in the deal. For example, if the hand is 4 apples, and there are only 3 apples in the deal, then this hand is not achievable.
4. Remove all hands that have a lower valuation than the current hand. The remaining set is the set of achievable goals.
5. Calculate a set of plans for each achievable goal. A plan is a sequence of trades (one item in hand for one item out of hand) that will lead to the goal. There are many possible plans for each goal. For simplicity, we ignore any plans that involve cycles, where the same hand appears more than once.
6. Calculate the expected utility (outcome) of each plan. Each plan will have a probability distribution of outcomes, based on the probability that each trade is successful. The outcome will be the hand that results from the end state, or the state before the trade that fails. For example, suppose the simulator's hand is (apple, apple, orange), and the simulator's plan is (apple→orange, orange→grape). The three possible outcomes are:

   (apple, orange, grape) (i.e., if the plan succeeds) the probability is calculated as $P(t1) * P(t2)$.

   (apple, orange, orange) (i.e., if the first trade succeeds and the second fails) the probability is calculated as $P(t1) * (1 - P(t2))$.

   (apple, apple, orange) (i.e., if the first trade fails) the probability is calculated as $1 - P(t1)$.

   Therefore, the simulator can calculate the expected utility of each plan, by multiplying the probability of each trade with the valuation of each hand from step 2. We set the probability of success of each trade to 0.5 (i.e., uninformative probability). This value of probability represents the fact that the simulator does not know a priori whether the trade will succeed or not.
7. Select the plan which has the highest expected utility as the plan that the policy will follow.
8. Select an action implementing the plan that was chosen in the previous step, as follows: if the plan is completed (i.e., the simulator reached the goal), the policy will select Keep as an action. If the plan is not completed and there is a pending offer which will allow the plan to move forward, the policy will select Accept as an action. Otherwise, the policy will select Offer as an action. The addressee of the offer is randomly selected from the traders holding the item which is required for moving the plan forward.

In addition to the above hand-crafted trader simulator's policy, we also use a random policy.

5 Evaluation

In this section, we evaluate the learner's policies learned with (1) different algorithms i.e., LinQ, LSPI, and NFQ (see Section 2), (2) different reward functions i.e., Equations 3 and 4 (see Section 4.1), and (3) different numbers of traders.

The evaluation is performed in trading dialogs with different numbers of participants (from 2 players to 4 players), and different trader simulator's policies (hand-crafted policy or random policy as presented in Section 4.2). More specifically, there are 9 different setups:

H: 2-party dialog, where the trader simulator follows a hand-crafted policy.

R: 2-party dialog, where the trader simulator follows a random policy.

HxH: 3-party dialog, where both trader simulators follow hand-crafted policies.

HxR: 3-party dialog, where one trader simulator follows a hand-crafted policy and the other one follows a random policy.

RxR: 3-party dialog, where both trader simulators follow random policies.

HxHxH: 4-party dialog, where all three trader simulators follow hand-crafted policies.

HxHxR: 4-party dialog, where two trader simulators follow hand-crafted policies and the other one follows a random policy.

HxRxR: 4-party dialog, where one trader simulator follows a hand-crafted policy and the other ones follow random policies.

RxRxR: 4-party dialog, where all three trader simulators follow random policies.

There are also 9 different learner policies:

AlwaysKeep: weak baseline which always passes the turn.

Random: weak baseline which randomly selects one action from all possible valid actions.

LinQ-End: learned policy using LinQ and reward given at the end of the dialog.

LSPI-End: learned policy using LSPI and reward given at the end of the dialog.

NFQ-End: learned policy using NFQ and reward given at the end of the dialog.

LinQ-Incr: learned policy using LinQ and an incremental reward.

LSPI-Incr: learned policy using LSPI and an incremental reward.

NFQ-Incr: learned policy using NFQ and an incremental reward.

Handcraft1: strong baseline following the handcrafted policy presented in Section 4.2.

Handcraft2: strong baseline similar to Handcraft1 except the plan is randomly selected from the set of plans produced by step 6, rather than picking only the highest utility one (see Section 4.2).

We use the Pybrain library (Schaul et al., 2010) for the RL algorithms LinQ, LSPI, and NFQ. The learning parameters follow the default Pybrain settings except for the discount factor γ; we set the discount factor γ to 1. We consider 2000 dialogs as one epoch, and learning is finished when the number of epochs becomes 200 (400,000 dialogs). The policy at the epoch where the average reward reaches its highest value is used in the evaluation.

We evaluate the learner's policy against trader simulators. We calculate the average reward of the learner's policy in 20000 dialogs. Furthermore, we show how fast the learned policies converge as a function of the number of epochs in training.

In terms of comparing the average rewards of policies (see Figure 1), NFQ-Incr achieves the best performance in almost every situation. In 2-party trading, the performance of NFQ-Incr is almost the same as that of Handcraft2 which achieves the best score, and better than the performance of Handcraft1. In both 3-party and 4-party trading, the performance of NFQ-Incr is better than that of the two strong baselines, and achieves the best score. In contrast to NFQ-Incr, the performance of the other learned policies is much worse than that of the two strong baselines. As the number of trader simulators who follow a random policy increases, the difference in performance between NFQ-Incr and the other learned policies tends to also increase. One reason is that, as the number of trader simulators who follow a random policy increases, the variability of dialog flow also increases. Trader simulators that follow a hand-crafted policy behave more strictly than trader simulators that follow a random policy. For example, if the trader simulator following a hand-crafted policy reaches its goal, then there is nothing else to do except for Keep. In contrast, if a trader simulator following a random policy reaches its goal, there is still a chance that it will accept an offer which will be beneficial to the learner. As a result there are more chances for the learner to gain better outcomes, when the complexity of the dialog is higher. In summary, our results show that combining NFQ with an incremental reward produces the best results.

Moreover, the learning curve in 2-party trading (Figure 2 in the Appendix) indicates that, basically, only the NFQ-Incr achieves stable learning. NFQ-Incr reaches its best performance from epoch 140 to epoch 190. On the other hand, LSPI somehow converges fast, but its performance is not so high. Moreover, LinQ converges in the first epoch, but it performs the worst.

6 Conclusion

In this paper, we used RL to learn the dialog system's (learner's) policy in a multi-party trading scenario. We experimented with different RL algorithms and reward functions. The negotiation policies of the learner were learned and evaluated through simulated dialog with trader simulators. We presented results for different numbers of traders. Our results showed that (1) even in simple multi-party trading dialog tasks, learning an effective negotiation policy is a very hard problem; and (2) the use of neural fitted Q iteration combined with an incremental reward function produces as effective or even better negotiation policies than the policies of two strong hand-crafted baselines.

For future work we will expand the dialog model to augment the dialog state with information about the estimated payoff matrix of other traders. This means expanding from an MDP-

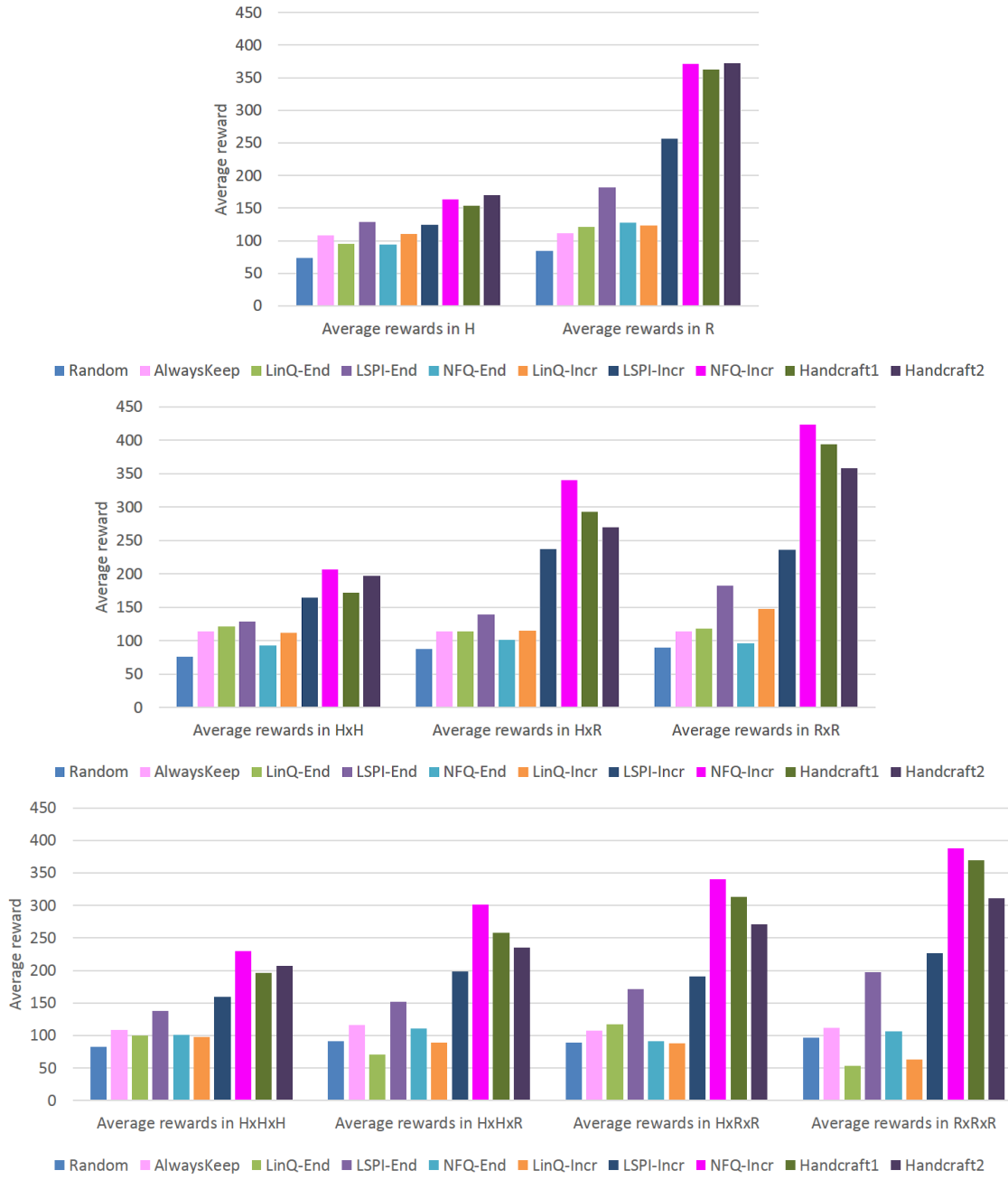

Figure 1: Comparison of RL algorithms and types of reward functions. The upper figure corresponds to 2-party dialog, the middle figure to 3-party dialog, and the lower figure to 4-party dialog. In these figures, the performances of the policies are evaluated by using the reward function given by Equation 3.

based dialog model to a POMDP-based model. We will also apply multi-agent RL (Georgila et al., 2014) to multi-party trading dialog. Furthermore, we will perform evaluation with human traders. Finally, we will collect and analyze data from human trading dialogs in order to improve our models and make them more realistic.

Acknowledgments

This research was partially supported by the 2014 Global Initiatives Program, JSPS KAKENHI Grant Number 24240032, and the Commissioned Research of the National Institute of Information and Communications Technology (NICT), Japan. This material was also based in part upon work supported by the National Science Foundation under Grant Number IIS-1450656, and the U.S. Army. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the United States Government, and no official endorsement should be inferred.

References

Nicholas Asher and Alex Lascarides. 2013. Strategic conversation. *Semantics and Pragmatics*, 6:2:1–62.

Layla El Asri, Romain Laroche, and Olivier Pietquin. 2013. Reward shaping for statistical optimisation of dialogue management. *In Proc. of SLSP*.

Ioannis Efstathiou and Oliver Lemon. 2014. Learning non-cooperative dialogue behaviours. In *Proc. of SIGDIAL*.

Kallirroi Georgila and David Traum. 2011. Reinforcement learning of argumentation dialogue policies in negotiation. In *Proc. of INTERSPEECH*.

Kallirroi Georgila, Maria K. Wolters, and Johanna D. Moore. 2010. Learning dialogue strategies from older and younger simulated users. In *Proc. of SIGDIAL*.

Kallirroi Georgila, Claire Nelson, and David Traum. 2014. Single-agent vs. multi-agent techniques for concurrent reinforcement learning of negotiation dialogue policies. In *Proc. of ACL*.

Kallirroi Georgila. 2013. Reinforcement learning of two-issue negotiation dialogue policies. In *Proc. of SIGDIAL*.

Markus Guhe and Alex Lascarides. 2012. Trading in a multiplayer board game: Towards an analysis of non-cooperative dialogue. In *Proc. of CogSci*.

Peter A. Heeman. 2009. Representing the reinforcement learning state in a negotiation dialogue. In *Proc. of ASRU*.

Takuya Hiraoka, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2014. Reinforcement learning of cooperative persuasive dialogue policies using framing. In *Proc. of COLING*.

Michail G. Lagoudakis and Ronald Parr. 2003. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149.

Esther Levin, Roberto Pieraccini, and Wieland Eckert. 2000. A stochastic model of human-machine interaction for learning dialog strategies. In *Proc. of ICASSP*.

Yun Mayya, Lee Tae Kyung, and Ko Il Seok. 2011. Negotiation and persuasion approach using reinforcement learning technique on broker's board agent system. In *Proc. of IJACT*.

Amy McGovern and Andrew G. Barto. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proc. of ICML*.

Andew Y. Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. *In Proc. of ICML*.

Elnaz Nouri, Kallirroi Georgila, and David Traum. 2012. A cultural decision-making model for negotiation based on inverse reinforcement learning. In *Proc. of CogSci*.

Elnaz Nouri, Sunghyun Park, Stefan Scherer, Jonathan Gratch, Peter Carnevale, Louis-Philippe Morency, and David Traum. 2013. Prediction of strategy and outcome as negotiation unfolds by using basic verbal and behavioral features. In *Proc. of INTERSPEECH*.

Praveen Paruchuri, Nilanjan Chakraborty, Roie Zivan, Katia Sycara, Miroslav Dudik, and Geoff Gordon. 2009. POMDP based negotiation modeling. In *Proc. of MICON*.

Michael Pfeiffer. 2004. Reinforcement learning of strategies for Settlers of Catan. In *Proc. of the International Conference on Computer Games: Artificial Intelligence, Design and Education*.

Martin Riedmiller. 2005. Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method. In *Proc. of ECML*.

Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. 2010. Pybrain. *The Journal of Machine Learning Research*, 11:743–746.

Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement learning: An introduction*. MIT Press.

David Traum. 2008. Extended abstract: Computational models of non-cooperative dialogue. *In Proc. of SEMDIAL-LONDIAL*.

Marilyn A. Walker, Diane J. Litman, Candace A. Kamm, and Alicia Abella. 1998. Evaluating spoken dialogue agents with PARADISE: Two case studies. *Computer Speech and Language*, 12(4):317–347.

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

Jason D. Williams and Steve Young. 2007a. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech and Language*, 21(2):393–422.

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

Yi Zou, Wenjie Zhan, and Yuan Shao. 2014. Evolution with reinforcement learning in negotiation. *PLoS One*, 9(7).

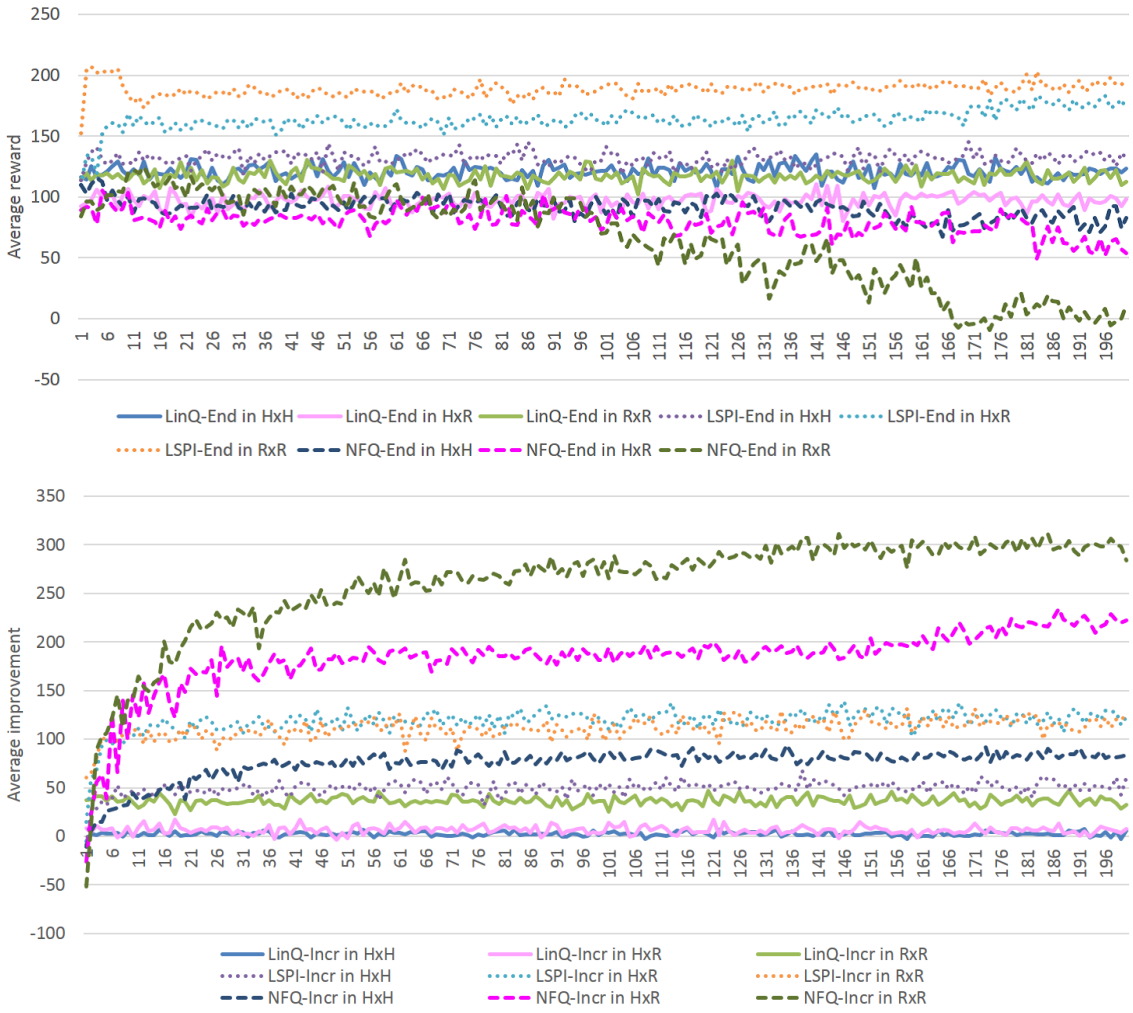

Figure 2: Number of epochs vs. performance of learned policies in 2-party trading. The upper figure shows the performance when the reward is given by Equation 3. The lower figure shows the performance when the reward is given by Equation 4.