# Revisiting user simulation in dialogue systems : do we still need them ? : will imitation play the role of simulation ?

Senthilkumar Chandramohan

# Revisiting User Simulation in Dialogue Systems.
# Do we still need them?
# Will imitation play the role of simulation?

Senthilkumar Chandramohan

Machine Learning and Interactive Systems group, Supélec - Metz

Laboratoire Informatique d'Avignon

Université d'Avignon et des Pays de Vaucluse

A thesis submitted for the degree of

*Doctor of Philosophy*

September 2012

Dedicated to my beloved parents.

# Acknowledgements

In the journey of life, numerous people walk in and out of our lives. Yet, at times we wish some of those who came into our lives stayed for ever. It is with that sense I am writing this note at the end of my work contract with Supélec. One of the best things that ever happened in my life is to have an opportunity to pursue my doctoral research at Supélec and work with two great minds in the form of Prof. Olivier Pietquin and Dr. Matthieu Geist. Things got even better when I registered my thesis at the Université d'Avignon and continued working under the guidance of Prof. Fabrice Lefèvre. Between 2009 and 2012, I had numerous opportunities to interact with them and learn a great deal of things. I am really grateful for having received their unlimited support at good times as well as those difficult days. I wish to convey my deepest gratitude to my advisers for their priceless time, efforts and above all their inspiration. Olivier, the trust you placed on me was more than what I had on myself at times.

Next, I would like to thank my fellow students Lucie, Edouard, Bassem, Billal, Constantinos, Vianney and Andreas for making the salle de doctorant a conducive and livelier place to work. As part of the IMS group at Supélec, it was a pleasure to work with Hervé, Stéphane, Michel, Frédéric, JB, Jérémy, Abolfazl and others such as Beate, Gilles. I thank the networking team for providing the necessary computational infrastructure. A special thanks to our secretaries Fabienne, Thérèse and Danielle. During my stay in Metz, the Supélec administration has been a great support in several aspects and so I would like convey my sincere thanks to the campus director Dr. Serge Perrine and the research director Dr. Joël Jacquet.

I also take this opportunity to thank Prof. Oliver Lemon for introducing me to the fascinating domain of spoken dialogue research. The time I spent in Edinburgh was primarily responsible for kindling my interest on spoken dialogue systems. I must thank Dr. Verena Rieser who in the first place was responsible for my interaction with Oliver. Special thanks to Srinivasan and Udhyakumar, who have been a great source of motivation right from my undergraduate days.

Finally, I would like to extend my deepest gratitude to my family. Without their continued support and wishes, I would not be in a position to complete my doctoral research. I sincerely thank Er. Sukumar Ramamoorthy, Mr. Elango Ramamoorthy and Mr. Arulkumar Sambandam for their motivation and guidance at different times in my life. I started my schooling under the auspicious of my lovable grandparents and since then I have received their endless affection. I am really grateful to these selfless and noble souls. I wish my grandma lived long enough to see this day.

I sincerely thank the Conseil Régional de Lorraine and the European Commission's FP7 (216594) Classic project for jointly financing my doctoral research. A special thanks to all my French friends for welcoming me with open hands and for giving numerous good moments to cherish.

Vive la France !!!

# Abstract

Recent advancements in the area of spoken language processing and the wide acceptance of portable devices, have attracted significant interest in spoken dialogue systems. These conversational systems are man-machine interfaces which use natural language (speech) as the medium of interaction. In order to conduct dialogues, computers must have the ability to decide when and what information has to be exchanged with the users. The dialogue management module is responsible to make these decisions so that the intended task (such as ticket booking or appointment scheduling) can be achieved. Effective functioning of a dialogue system depends on the quality of the strategy used for making these decisions. Thus learning a good strategy for dialogue management is a critical task.

In recent years reinforcement learning-based dialogue management optimization has evolved to be the state-of-the-art. A majority of the algorithms used for this purpose needs vast amounts of training data. However, data generation in the dialogue domain is an expensive and time consuming process. In order to cope with this and also to evaluate the learnt dialogue strategies, user modelling in dialogue systems was introduced. These models simulate real users in order to generate synthetic data. Being computational models, they introduce some degree of modelling errors. In spite of this, system designers are forced to employ user models due to the data requirement of conventional reinforcement learning algorithms.

As part of this manuscript, a set of sample efficient Approximate Dynamic Programming and Kalman Temporal Differences class of algorithms are adapted to dialogue optimization. Experimental results indicate that these algorithms are indeed sample efficient and can learn optimal dialogue strategies from limited amount of training data when compared to the conventional algorithms. As a consequence of this, user models are no longer required for the purpose of optimization, yet they continue to provide a fast and easy means for quantifying the quality of dialogue strategies. Since existing methods for user modelling are relatively less realistic compared to real user behaviors, the focus is shifted towards user modelling by means of inverse reinforcement learning. Using experimental results, the proposed method's ability to learn a computational models with real user like qualities is showcased as part of this work.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Spoken Dialogue Systems (SDS) are natural language interfaces for Human Computer Interaction (HCI). Speech is used as the medium of interaction. Some example scenarios for dialogue based interaction with computers are shown in Figure 1.1. Dialogue management engine of the SDS navigates it to accomplish a specific task such as flight booking [Williams & Young, 2007], town information [Lemon *et al.*, 2006] *etc.*

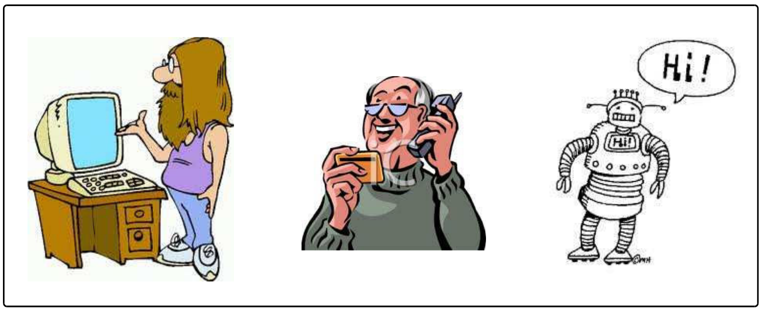


Figure 1.1: Spoken dialogue interface for human computer interaction

## 1.1 Problem statement

State-of-the-art dialogue systems [Lemon & Pietquin, 2007; Rieser & Lemon, 2011], rely on Reinforcement Learning (RL) [Sutton & Barto, 1998] based optimization schemes in order to learn an user adaptive and (noise) robust dialogue strategy. However, poor sample efficiency of conventional RL algorithms forces system designers to employ user simulations [Levin *et al.*, 2000]. User simulations are computational models used to generate synthetic dialogues. The primary problem in employing user simulation is twofold: (i) being a computational model in itself user simulators introduce modelling errors, (ii) existing approaches for building user simulators [Schatzmann *et al.*, 2006a] focus on replicating dialogue corpus rather than behaving like a real user. Dialogue optimization which involves such user simulators results in adapting[1] dialogue strategies towards some generic, non-existent users.

In order to solve this problem, *sample efficient approaches for dialogue optimization* are analysed as part of this manuscript. By means of experimental results, sample efficiency and effectiveness of Approximate Dynamic Programming (ADP) [Bellman & Dreyfus, 1959] and Kalman Temporal Differences (KTD) algorithms [Geist & Pietquin, 2010b] when applied to dialogue are validated [Pietquin *et al.*, 2011a,b]. The primary motivation behind this effort is to avoid using user simulators for dialogue optimization. However, optimized dialogue strategies still ought to be evaluated. Involving prospective users or domain experts to evaluate their performance is one possible solution. Yet, it is an expensive and time consuming process. Thus, user simulators continue to play a role at least for the sake of evaluation. Behavior of real users tends to be much more complex rather than merely responding to the current dialogue act. For example, every user response is often based on the progress of the dialogue and their goal. Also real users can generalize and adapt their behavior based on the dialogue manager they are interacting with. In order to overcome the shortcomings of existing methods for user modelling and also to make user simulation more realistic, a novel method based on *Inverse Reinforcement Learning* (IRL) [Chandramohan *et al.*, 2011a; Ng & Russell, 2000] is proposed. This method treats the task of user simulation as a sequential decision making problem. Imitation learning is performed to solve this problem and thereby simulate real user behavior.

[1] Quality of user simulation tends to have a direct impact on the quality of dialogue strategy.

## 1.2 Resulting contributions

The following is a list of contributions resulting from this thesis. As discussed in 1.1, this manuscript focuses on two key aspects of statistical dialogue management: (i) sample efficiency in dialogue optimization and (ii) IRL-based user simulation.

**Sample efficient dialogue optimization**

1. Chandramohan, S., Geist, M. & Pietquin, O. (2010a). Optimizing Spoken Dialogue Management with Fitted Value Iteration. In *Proc. of InterSpeech 2010*, Makuhari (Japan)

2. Chandramohan, S., Geist, M. & Pietquin, O. (2010b). Sparse Approximate Dynamic Programming for Dialog Management. In *Proc. of SIGDial*, Tokyo (Japan)

3. Daubigney, L., Gasic, M., Chandramohan, S., Geist, M., Pietquin, O. & Young, S. (2011). Uncertainty management for on-line optimisation of a POMDP-based large-scale spoken dialogue system. In *Proc. of Interspeech 2011*, 1301–1304, Florence (Italy)

4. Pietquin, O., Geist, M. & Chandramohan, S. (2011a). Sample Efficient On-line Learning of Optimal Dialogue Policies with Kalman Temporal Differences. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, Barcelona (Spain)

5. Pietquin, O., Geist, M., Chandramohan, S. & Frezza-Buet, H. (2011b). Sample-Efficient Batch Reinforcement Learning for Dialogue Management Optimization. *ACM Transactions on Speech and Language Processing*, **7**, 7:1–7:21

**Inverse reinforcement learning based user simulation**

6. Chandramohan, S., Geist, M., Lefèvre, F. & Pietquin, O. (2011a). User Simulation in Dialogue Systems using Inverse Reinforcement Learning. In *Proc. of Interspeech 2011*, Florence (Italy)

7. Chandramohan, S., Geist, M., Lefèvre, F. & Pietquin, O. (2012b). Clustering Behaviors of Spoken Dialogue Systems Users. In *Proc. of the 37th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2012)*, Kyoto (Japan)

8. Chandramohan, S., Geist, M., Lefèvre, F. & Pietquin, O. (2012c). Regroupement non-supervisé d'utilisateurs par leur comportement pour les systèmes de dialogue parlé. In *Journées Francophones de Planification, Décision et Apprentissage pour la conduite de systèmes (JFPDA 2012)*, Nancy (France)

9. Chandramohan, S., Geist, M. & Pietquin, O. (2011b). Apprentissage par Renforcement Inverse pour la Simulation d'Utilisateurs dans les Systèmes de Dialogue. In *Sixièmes Journées Francophones de Planification, Décision et Apprentissage pour la conduite de systèmes (JFPDA 2011)*, 7 pages, Rouen (France)

10. Chandramohan, S., Geist, M., Lefèvre, F. & Pietquin, O. (2012a). Behavior Specific User Simulation in Spoken Dialogue Systems. In *Proc. of the IEEE ITG Conference on Speech Communication (to appear)*, Braunschweig, Germany

11. Keizer, S., Rossignol, S., Chandramohan, S. & Pietquin, O. (2012). User Simulation in the Development of Statistical Spoken Dialogue Systems. In *Data driven methods for Adaptive Spoken Dialogue Systems*, Springer-Verlag New York Inc (To appear), (To appear)

## 1.3 Thesis layout

This manuscript is organized in three parts: Part I, Part II and Part III. The basic idea in such an organization is to first introduce the necessary foundations and then outline the resulting contributions. To begin with Part I presents an overview on statistical dialogue management. It includes three chapters: Chapter 2 introduces spoken dialogue management and Chapter 3 outlines machine learning schemes for dialogue optimization and finally, Chapter 4 formally introduces user simulations in dialogue systems. Part II outlines the resulting contributions with regard to sample efficient dialogue optimization. This part is organized in two chapters: Chapter 5 studies the effects of dialogue optimization using ADP and Chapter 6 outlines the effectiveness of KTD-based algorithms when applied to dialogue optimization. Eventually, Part III summarizes the contributions with regard to the proposed IRL-based user simulation. The final part of this manuscript is organized in to three chapters: Chapter 7 presents a short overview on IRL and user simulation using IRL. Chapter 8 studies the possibility to cluster behaviors of dialogue system users and Chapter 9 presents an overview on co-adaptation in dialogue systems.

The rest of this section presents a detailed description on the layout of this manuscript. Chapter 2 provides an overview on dialogue systems and statistical dialogue management. To begin with the architecture of a dialogue system is outlined in 2.1, following which the dialogue management task is formally defined in 2.2. Dialogue state and state transitions are introduced in 2.2.1. Introduction to the dialogue policy along with an explanation on its role in dialogue management are presented in 2.2.2. A short summary on the taxonomy of dialogue systems is presented in 2.2.3. Challenges to be addressed in the process of dialogue management in case of man-machine interaction are outlined in 2.2.4. State-of-the-art in statistical dialogue management is summarized in 2.3. Chapter 3 is dedicated to provide an overview on Markov Decision Processes (MDP) [Puterman, 1994] and associated optimization techniques. Section 3.1 formally defines an MDP and 3.2 introduces the Bellman equations and explains how it can be used to solve MDPs. Dynamic programming (DP) [Bellman, 1957a] based algorithms for solving MDPs are introduced in 3.3. An overview on RL-based solution techniques for solving MDPs is presented in 3.4. Eventually dialogue management using MDPs is discussed in 3.5.

Role of user simulation in SDS is discussed in Chapter 4. To begin with 4.1 introduces user simulation in SDS, following which an overview on some of the existing approaches for building user simulators are described. Section 4.2 lists commonly used metrics for evaluating the performance of user simulators. Section 4.3 outlines few shortcomings in present day user simulation techniques.

Chapter 5 introduces ADP algorithms and studies their effectiveness when applied for dialogue optimization. To begin with value function approximation is outlined in 5.1, following which ADP-based Fitted Value Iteration is discussed in 5.2. Automatic feature selection scheme for value function approximation and Sparse-ADP algorithms are discussed in 5.4, 5.5 and 5.6 respectively. Eventually, experimental results on dialogue optimization using ADP and sparse ADP algorithms are presented in 5.7

An introduction to KTD framework and dialogue optimization using KTD-$Q$ and KTD-SARSA are presented in Chapter 6. Policy optimization using online/off-policy are discussed in 6.1 ($Q$-learning) and 6.2 (KTD-$Q$), following which policy optimization using online/on-policy algorithms are outlined in 6.3 (SARSA and KTD-SARSA). Eventually 6.5 outlines the effectiveness of these algorithms when applied for dialogue optimization. Given the shortcomings of existing methods for user simulation, Chapter 7 presents a novel method for user simulation using IRL. The equivalence of the user simulation task to a sequential decision making problem is discussed in 7.1. IRL problem elicitation and description of IRL-based user simulation are presented in 7.2. Eventually 7.3 outlines the experimental set-up and analyse the results.

Chapter 8 describes how behavior clustering can be performed using feature expectations. To begin with the proposed scheme for clustering is outlined in 8.1. Once similar behaviors are identified and clustered, specific behaviors can then be imitated using IRL-based user simulation. Eventually 8.2 outlines a set of experiments carried out to validate the effectiveness of the proposed methods. Chapter 9 introduces the co-adaptation framework in dialogue systems and studies its impact towards dialogue evolution. Natural forms co-adaptation that occurs in human-human communication are discussed in 9.1. The proposed scheme for co-adaptation in dialogue systems is discussed in 9.2. Section 9.3 describes the experimental set-up and analyses the consequence of co-adaptation for a 2-slot restaurant information SDS.

Finally, Chapter 10 revisits the experimental results and derives a conclusion on the proposed algorithms and methods for dialogue optimization and user modelling. To begin with 10.1 derives a conclusion on the effectiveness of ADP and KTD algorithms for dialogue optimization. Following which 10.2 summarizes the effects of user modelling using IRL. Section 10.3 revisits some of the advantages of treating the user modelling task as a sequential decision making problem and solving it using IRL. Eventually,  10.4 identifies and outlines some directions of future works.

# Part I

# Statistical Dialogue Management

# Chapter 2

# Spoken Dialogue Systems

Human Computer Interaction (HCI) is a field of study which focuses on designing interfaces that facilitate the interaction between computers and their prospective users. Ever since computers began to play a key role in our day-to-day lives, interfaces such as keyboards, mouse enabled graphical layouts and the more recent touch screens have all become widely accepted means for HCI. However, in order to use these devices, the users are expected to have some amount of knowledge on their functionality and how they can be used. Unlike such interfaces, dialogue systems provide a more natural mean for HCI using speech as the medium of interaction. Thus users can now converse with computers just like in human-human interaction. Dialogue based interaction means now computer enabled services such as weather forecast or flight information can be received remotely (for instance, over telephone).

Looking at human-human conversation in a psychological perspective: we humans tend to involve in a dialogue in order to convey or receive information. At granular level this task is accomplished by means of utterances (speech based utterances or non-verbal utterances such as facial expressions, body language *etc*). Performative utterances in the form of speech acts convey a short piece of information. This information is interpreted based on the context of the dialogue. The dialogue context is composed of a sequence of utterances (uttered during the history of the communication). Thus to summarize, parties involved in communication alternately generate communicative utterances or speech acts so that the intended task (of conveying or seeking some information) can be achieved over a period of time. More detailed information on psychological aspects of human-human communication can be found in [Austin, 1975]. The task of choosing a sequence of performative utterances or speech acts[1] and maintaining as well as updating the dialogue history or context can be termed as *spoken dialogue management*. We humans acquire the necessary skills to conduct and manage a dialogue as we grow up.

In order to conduct dialogue with human users, computers should have the ability to choose and perform a sequence of speech acts as well as maintain the dialogue context so that speech acts generated by users can be suitably interpreted. Tasks to be accomplished by a dialogue system may range from simple tasks (such as flight booking, tourist information) to complex tasks (such as tutoring). The following chapter is organized as follows: To begin with an overview on the architecture of SDS is presented in Section 2.1. Computer enabled spoken dialogue management is elicited in detail in Section 2.2. Section 2.2.1 explains how dialogue progress can be observed in the form of dialogue context and how it can be represented using the Information State (IS) paradigm. The significance and the structure of dialogue management policy is outlined in Section 2.2.2. A short note on the taxonomy of dialogue systems is presented in Section 2.2.3. Challenges to be addressed in the process of dialogue management in case of man-machine interaction are outlined in Section 2.2.4. Finally, Section 2.3 surveys the state-of-the-art statistical dialogue managers.

## 2.1 Architecture of dialogue systems

SDS are complex systems, which include multiple internal modules (as shown in Figure 2.1). The speech signal uttered by the user is converted into text by the Automatic Speech Recognition (ASR) module [Rabiner & Juang, 1993]. Natural Language understanding (NLU) module [Allen, 1995] then extracts the meaning of this text. The extracted information is passed on to the dialogue manager. In case of a multi-modal SDS (which for example employs a graphical user interface) users can choose to provide a non-verbal utterance. In the later case, the

[1] Every speech act is chosen in accordance to the dialogue context so that some task (such as train ticket booking or trouble shooting) can be achieved over a period of time.

dialogue manager is updated by both the language understanding module and event manager of the graphical user interface.

The dialogue manager [Frampton & Lemon, 2009; Levin & Pieraccini, 1998; Roy *et al.*, 2000] plays a key role in navigating the SDS to accomplish the specified task. Based on the user utterance it decides the next speech act to be performed by the SDS. Language generation module [Reiter & Dale, 2000] transforms the speech act of the SDS into a natural language text (with the help of a suitable language model). The text generated is presented to the user with a text-to-speech synthesis engine [Dutoit, 1997]. In case of a multi-modal SDS, a graphical user interface as shown in Figure 2.1 can be used to present the information.

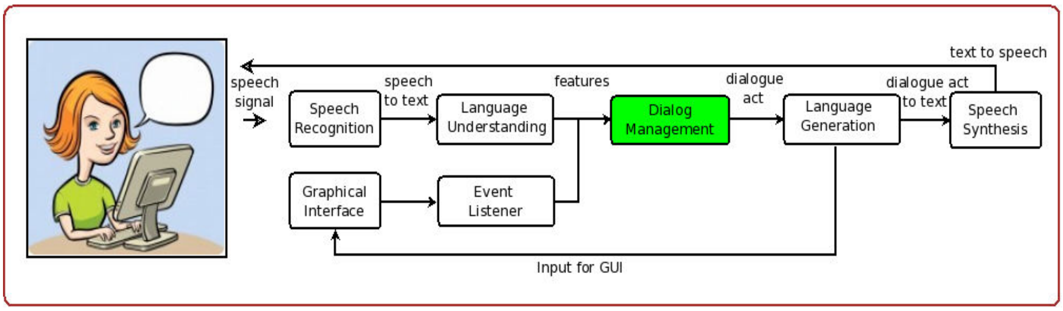


Figure 2.1: Architecture of a multi-modal spoken dialogue system

## 2.2 Spoken dialogue management

Dialogue management in itself is a problem which involves choosing and performing a sequence of speech acts. By means of step-by-step instructions, the dialogue manager governs the functioning of the SDS. Speech acts performed by the dialogue manager are termed as *dialogue acts*, while user's speech acts are termed as *user acts*. A dialogue turn constitutes of a dialogue act and an user act. At every dialogue turn, the dialogue manager intends either to seek information from the user or furnish information to the user. The choice of a dialogue act is based on the dialogue context. The primary aim of the SDS and hence the dialogue manager is to achieve its goal, taking user satisfaction into account.

Following are fictitious dialogues between a tourist guide and a tourist who is visiting some city. Let us assume that the tourist intends to eat in a cheap Italian restaurant located in the city-centre, but has no information about restaurants in the city. Thus to find a suitable restaurant he/she contacts the tourist guide at the information desk who has a database of restaurants. In order to serve the tourist *i.e.*, to search the database and refine the results, the guide has to know the tourist's preferences for certain fields or variables such as: (i) restaurant location, (ii) price-range, (iii) cuisine type, *etc.*

### Example Dialogue: 1

Guide: Hi, how can I help you?

Tourist: I am looking for a restaurant.

Guide: In which part of the city are you looking for a restaurant?

Tourist: City-centre please.

Guide: In what price-range are you looking?

Tourist: A cheap restaurant which serves Italian food.

Guide: Inform possible options {1,2,3...}

Tourist: Thank you

Guide: Good bye

### Example Dialogue: 2

Guide: Hi, how can I help you?

Tourist: I am looking for a cheap Italian restaurant in the city-centre.

Guide: Inform possible options {1,2,3...}

Tourist: Thank you

Guide: Good bye

*Example dialogues from the restaurant information domain*

Example dialogue 1 showcases the behavior of a tourist who has limited experience in interacting with the tourist guide. So when asked *how can I help you? or In which part of the city are you looking for a restaurant?* the tourist answers the questions directly and provide limited information. As the dialogue progresses the user grows in confidence and tends to provide more information as shown by his/her response for *In what price-range are you looking?* It can be observed that the response for this question includes information on the price-range of the restaurant being searched for (*i.e., cheap*) along with information on the type of the restaurant (*i.e., Italian*). However, example dialogue 2 showcases the behavior of the tourist who knows all the information required by the guide (perhaps as result of frequent interaction). Thus, when asked *how can I help you?*, the response is comprehensive and includes all information required by the guide (*i.e.,* price-range, location, restaurant-type).

Let us now assume that the role of the tourist guide has to be automated and a dialogue system will be employed in order to help the tourists. For the sake of simplicity let us assume that the dialogue system can only provide restaurant related information. In case of such a dialogue system, the dialogue manager's role is to seek information about the restaurant preferences and furnish the results from database. However, dialogue management is a challenging problem due to (i) the variations in user behaviors and (ii) the uncertainty due to speech recognition errors and/or language understanding errors. Based on the examples studied here it can be assumed that: the users of the dialogue system will have behaviors similar to the one showcased either in the example dialogue 1 or in the example dialogue 2. Thus it is safe to mention that the prospective users may have no prior knowledge or some (partial or complete) knowledge on what information will be requested by the system. More importantly such behaviors can be observed only during the time of interaction with the potential users. Often it is impossible to guess the user knowledge or say how the user will behave before actually initiating the dialogue. Thus dialogue systems should have the ability to cope and adapt to the variations in user behaviors. More information on challenges associated with dialogue management are discussed in Section 2.2.4.

### 2.2.1 Information state update

In case of a human-human dialogue, real user's ability to maintain and update the dialogue context comes in handy for dialogue management. Some internal measure of dialogue progress is made available naturally. In order to give this ability for computers, there is a necessity to track the progress of the dialogue. This can be achieved by the so called *dialogue state or context*. It is an internal measure of the dialogue progress and everything that can help the dialogue manager in choosing a suitable dialogue act. It compactly summarizes the dialogue history. At every turn, the dialogue manager updates the current dialogue context based on the *observed*[1] user utterance. The updated context is used to choose the next dialogue act. This process is repeated until the dialogue task is achieved by the system or the user terminates the dialogue (for instance, hangs-up the phone).

A simple, yet effective way for representing the dialogue context is to use the so called Information State (IS) paradigm [Larsson & Traum, 2000]. The dialogue context in itself consists of features of great relevance to the dialogue problem being treated. At each dialogue turn, based on the user response one or more features in the dialogue state are updated. The restaurant information dialogue problem studied here is essentially a form filling or slot filling dialogue task. In order to search the database, the SDS must know the user preferences: (i) whether the user is looking for a restaurant or a bar, (ii) in which part of the city the user is looking for the restaurant, (iii) in what price-range and finally (iv) what type of restaurant the user is looking for, *i.e.,* whether he is looking for an Italian or an Indian restaurant. It is also necessary to track dialogue events such as: (v) whether the user has been greeted or not, (vi) whether the results retrieved from the database have been informed to the user or not.

Since these (six) features correspond to the user preferences and the dialogue progress, they play a key role during the dialogue. Thus while designing a dialogue

[1] Observed user utterance is indeed the user utterance subject to speech recognition and language understanding errors. The utterance recognized by the speech recognition engine and interpreted by the language understanding module may not always be the actual user utterance, primarily due to the possibility of errors (refer Section 2.2.4).

manager it is necessary to include them in the dialogue state. Features of the dialogue state can be defined at the word level or at the intention level. For example, a slot which corresponds to restaurant-type can take values such as Italian, Mexican, Indian *etc.* In case of word level feature definition the slot can have values: (i) 0 corresponds to Italian, (ii) 1 corresponds to Mexican, (iii) 2 corresponds to Indian, *etc.* However, in case of intention level definition the slot can only take two values: (i) 0 corresponds to a scenario where the user is yet to provide the value, (ii) 1 corresponds to a scenario where the user has already provided the value. Defining the feature at intention level provides a compact way for representing the dialogue problem. Thus dialogue state features are defined at the intention level in the rest of this manuscript. While defining the features at the intention level, it is also possible to have continuous values rather than binary values as discussed here[1]. To summarize the restaurant information SDS will have the following binary features in its state representation:

$\phi_1$ : search-id (sample values: restaurant or bar)

$\phi_2$ : location (sample values: city centre or market square)

$\phi_3$ : price-range (sample values: expensive or moderate or cheap)

$\phi_4$ : type (sample values: Italian restaurant or jazz bar)

$\phi_5$ : greet-user

$\phi_6$ : inform-user

*Restaurant information SDS: example of dialogue state features*

Thus, the dialogue state takes the form: $\{\phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6\}$, where all the features are binary variables. When the dialogue episodes begins (i.e. the initial state - {0, 0, 0, 0, 0, 0}) all the features are set to zero. Once the dialogue manager chooses and performs a dialogue act $a$, the user responds with a suitable

[1] It may be useful to note that state of the art statistical speech recognition engines apart from providing the recognized text, also provide a measure of recognition confidence. Thus, a feature can be defined to have continuous values which correspond to the ASR confidence score of a slot-value (refer Section 5.7.1 for an example scenario)

user act. A possible list of dialogue acts that can be performed by the dialogue manager is listed below:

1. ask-destination-type
2. ask-restaurant-type
3. ask-restaurant-location
4. ask-restaurant-price-range
5. inform-restaurants
6. greet-user
7. close-dialogue

*Restaurant information SDS: list of dialogue acts*

Based on the current dialogue state $s$, the dialogue act $a$ and the *observed* user utterance, state transition to a new state $s'$ occurs. Following is an example for dialogue state transition. In this example since the user has been greeted feature $\phi_5$ of $s'$ is set to 1. Also since the user informs he is looking for a restaurant feature $\phi_1$ of $s'$ is set to 1.

**Dialogue turn:**

Guide: Hi, how can I help you?

Tourist: I am looking for a restaurant.

**Dialogue State Transition:**

Initial dialogue state ($s$): $\{0, 0, 0, 0, 0, 0\}$

Dialogue act ($a$): greet-user

User utterance: inform-restaurant

Next dialogue state ($s'$): $\{1, 0, 0, 0, 1, 0\}$

*Restaurant information SDS: state transition example*

| Current dialogue state | Dialogue act to be performed |
|---|---|
| {0, 0, 0, 0, 0, 0} | greet-user |
| {1, 0, 0, 0, 0, 0} | ask-destination-type |
| ⋮ | ⋮ |
| {1, 1, 1, 1, 1, 0} | inform-restaurants |
| {1, 1, 1, 1, 1, 1} | close-dialogue |

Table 2.1: Restaurant information SDS: dialogue policy structure

### 2.2.2 Dialogue management policy

In Section 2.2, the dialogue management task was defined as one in which the dialogue manager has to choose (decide which is the best action) and perform a sequence of actions in order to accomplish a specific task. Most often such decisions are made using the so called dialogue policy. In simple terms, given a dialogue state, the policy dictates what action should be performed next. We know that the dialogue state for the restaurant information dialogue problem includes six binary features. In total there will be 64 states (in other words $|S|$ is $2^6 = 64$). There will be a total of 448 (*i.e.* 64*7) state-action pairs $(s, a)$. The dialogue policy here can take a tabular format as shown in Table 2.1.

### 2.2.3 Taxonomy of dialogue systems

In general commercial dialogue systems can be classified into several groups based on the type of the dialogue task addressed by them. A partial list of different SDS types includes: (i) form filling dialogues (for example, town information [Lemon *et al.*, 2006; Seneff *et al.*, 1998], flight booking [Williams & Young, 2007]), (ii) tutorial dialogues (for example, natural language tutoring SDS [Vanlehn *et al.*, 2007]), (iii) virtual companions or assistants [Wilks, 2004]. Based on the strategy applied for dialogue management, SDS can further be classified into two types: (i) rule based systems and (ii) plan based systems. Rule based dialogue systems are governed by a set of rules (similar to grammar which governs the correctness of a sentence) [Jnsson, 1993]. The primary focus of these rules is to handle dialogues at the level of transitions. Plan based dialogue systems are governed by detailed plans which include information such as user intentions and the final task to be accomplished. Here the focus is to handle dialogues at the level of trajectories. Based on who is performing the lead role in the dialogue, SDS can be classified into three types: (i) system-initiative (where the dialogue manager conducts the dialogue), (ii) user-initiative (where the user conducts the dialogue) and (iii) mixed-initiative [Ferguson *et al.*, 1996] (where both the dialogue manager and the user can take turns to conduct the dialogue and need not wait for the other party to seek/furnish information).

Dialogue policy-based spoken dialogue management introduced so-far in this chapter can be perceived (to some extent) as a plan based SDS. There exists several possible ways to determine this policy. For instance, it is possible to manually specify a policy which defines what action to be performed given the current dialogue context. However, hand-crafting such a policy is a time consuming task even for small dialogue problems like the one discussed here. *Dialogue optimization* is the process of retrieving an optimal[1] dialogue policy. The goal of dialogue optimization process is to determine a policy that achieves the dialogue task in an efficient way. A key focus of research in the dialogue domain since last decade is to explore the possibility for dialogue optimization using the available dialogue corpora.

### 2.2.4 Why dialogue management is challenging?

As mentioned earlier, the dialogue manager updates the dialogue state based on the *observed* user utterance. Speech recognition engine and natural language understanding modules play a key role in observing the user utterances. However, there is a possibility for incorrect recognition (due to ASR channel noise) or misinterpretation (due to NLU) of user utterances. Such errors are propagated to the dialogue manager resulting in state transitions which were not intended by the user. Let us have a look at the following example which showcases the effect of error in observing user utterances. In this case the tourist is looking for a restaurant in the High-street. However, if an error does occur (as shown below), the user utterance is interpreted or understood in a wrong sense. Using an incorrect

[1] What makes a policy optimal and how it can be estimated are discussed in chapter 3

observation, the dialogue manager updates its state. Here the dialogue manager believes the user has provided information about price-range of the restaurant (whereas the user expects it to provide restaurant options in the high-street). It takes several dialogue turns for the dialogue manager to realize the mistake. Thus, it is often hard to recover from such errors. Yet, there always exists some amount of uncertainty in observing the user utterance. Since the dialogue manager relies heavily on the ASR and NLU modules, it should have the ability to cope with recognition errors and/or language understanding errors. This is often a key challenge in designing dialogue managers. It is important to note that as the complexity of the dialogue problem increases, chances for speech recognition errors (since the vocabulary becomes very large in size) and language understanding errors (since it is difficult to determine the correct semantic annotation: what information is provided under which scenario) increase exponentially.

**Dialogue turn:**

Guide: Hi, how can I help you?

Tourist: I am looking for a restaurant in the high-street.

**Dialogue State Transition:**

Initial dialogue state ($s$): $\{0, 0, 0, 0, 0, 0\}$

Dialogue act ($a$): greet-user

Actual user utterance: inform-restaurant, inform-location

Observed user utterance: inform-restaurant, inform-price-range

Next dialogue state ($s'$): $\{1, 0, 1, 0, 1, 0\}$

*Restaurant information SDS: impact of speech recognition errors*

One other challenge with regard to dialogue management, is caused by variations in user behaviors (recollect the variation in user behavior in dialogue examples 1 and 2). Often while interacting with users, a dialogue system encounters users with different behaviors. Some users may have frequently used the dialogue system in the past and thus often provide more information in every dialogue turn. Whereas some users may never have used the dialogue system and thus wait for the dialogue system to take the lead role. Since prospective user behaviors are different from each other, there is a pressing necessity to have an adaptive dialogue management policy.

## 2.3 State-of-the-art in dialogue systems

In order to cope with the challenges involved in dialogue management and also to provide a satisfying experience for prospective users, various components of the state-of-the-art commercial SDS are built using statistical data driven methods. The primary focus here is to build dialogue systems which are (i) user adaptive, (ii) robust to errors and (iii) capable of producing natural human-human like dialogues. The following section presents an overview on the state-of-the-art in dialogue systems.

### 2.3.1 Statistical user act interpretation

Statistical approaches for speech recognition [Jelinek, 1998] and language understanding [Hahn *et al.*, 2011; Lefèvre, 2007] have taken precedence in recent years. A vast majority of present day ASR and NLU modules apart from interpreting the user act, also provide some probabilistic score indicating their confidence on the interpretation. The basic idea here is to provide an opportunity for the dialogue management engine to cope with the associated uncertainty. Using this confidence scores, dialogue managers can choose to use an interpretation hypothesis as it is (in case of high confidence) or else choose to request the user to repeat the speech act (in case of very low confidence) or seek the user to confirm whether the dialogue system's interpretation is in line with the user goal (in case of intermediate confidence). State-of-the-art statistical ASR and NLU modules provide multiple hypothesis along with their corresponding confidence scores. This in turn helps dialogue management engines to maintain multiple hypothesis [Henderson & Lemon, 2008; Williams & Young, 2007] about user intentions or user goal. One of the key advantages in maintaining multiple hypothesis is the ability and the ease with which one hypothesis can be switched with another. More sophisticated formalisms of user act interpretation does exist for the purpose of dialogue management. For instance, [Pinault & Lefèvre, 2011] proposes a scheme in which user utterance interpretation in the form of semantic graphs is directly mapped to the summary space of a dialogue manager.

### 2.3.2 Dialogue policy optimization

As the complexity of the dialogue problem increases, the size of the state-action space also increase exponentially[1]. Dealing with large state-space problems manually is a tedious or even an impossible task. Statistical dialogue optimization approaches aim at retrieving the dialogue policy from the dialogue corpus or by interacting with users (or user simulators). Dialogue management problem (as explained in Section 2.2), involves making a sequence of decisions. Given its equivalence to the sequential decision making problem, it is often modelled as a Markov Decision Process (MDP) [Bellman, 1957b; Levin & Pieraccini, 1998] or a Partially Observable Markov Decision Process (POMDP) [Sondik, 1978; Williams & Young, 2007].

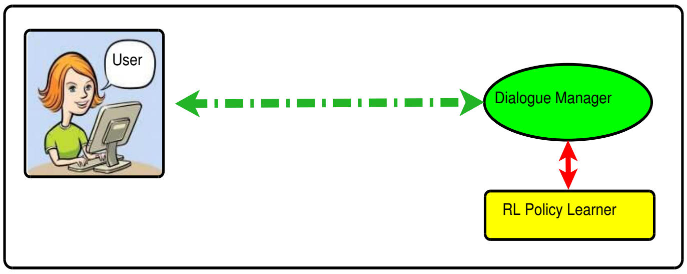


Figure 2.2: Dialogue optimization involving real users

Once the dialogue problem is casted as an MDP (as discussed in Section 3.5) or POMDP dialogue policy optimization can be performed using Reinforcement Learning (RL) [Kaelbling *et al.*, 1996; Sutton & Barto, 1998]. There exists a wide range of RL algorithms (as discussed in Section 3.4). One possibility to

[1] As the complexity of the dialogue problem increases, more and more features are required to define the dialogue state and also the list of possible dialogue acts increases. Thus, resulting in exponential increase in the size of state-action space.

perform this optimization indeed is to learn from live interactions with real users. Dialogue optimization in an online RL set-up is showcased in Figure 2.2.

### 2.3.3 User simulations in dialogue systems

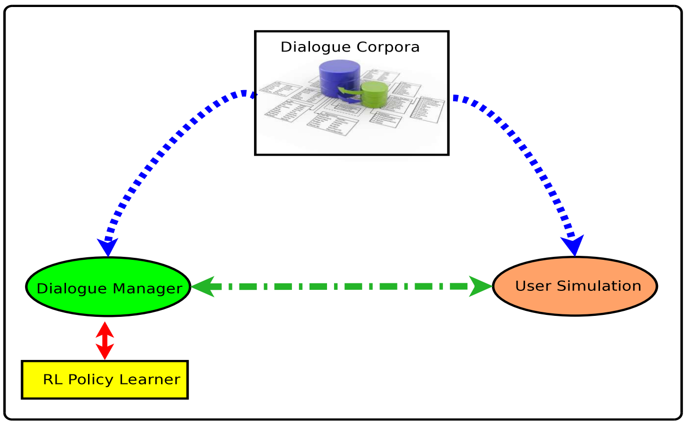


Figure 2.3: Dialogue optimization involving user simulation

Data (dialogue corpora) used for dialogue optimization is obtained from real users. It generally consists of a set of human-human or human-machine dialogues. If such dialogues are not available, dialogue corpora can be generated using a Wizard-of-Oz[1] set-up [Rieser, 2008]. Standard RL algorithms used for policy optimization require vast amounts of data to converge. However, dialogue corpora generation and annotation is a time consuming and expensive process. To cope with the data requirement of RL algorithms, user modelling for dialogue simulation has received significant interest in the last decade [Levin *et al.*, 2000; Pietquin, 2006; Schatzmann *et al.*, 2006a].

User simulators are computational models used to simulate the behavior(s) of real users and thereby generate synthetic dialogues. Often user simulators are built from the available dialogue corpus. Synthetic dialogues generated with user simulators are utilized to perform dialogue optimization. Given its cost

[1] In which case real users interact (in order to generate a set of dialogues) with a wizard assuming or believing it to be a machine. The role of the wizard is often enacted by a human operator or the system designer.

effectiveness, user simulators are also employed to evaluate the quality of learned dialogue policies. However, user simulators introduce new sources of modelling errors. The quality of the dialogue policy directly depends on the quality of user simulator used for policy optimization [Schatzmann *et al.*, 2005]. Chapter 4 provides a short overview on some of the existing methods for user simulation. A more detailed information on such methods can be found in [Schatzmann *et al.*, 2006b].

### 2.3.4 Language generation in dialogue systems

As mentioned earlier, language generation modules are used to generate natural language text given the current dialogue act. They play a key role in determining the naturalness of human-machine dialogues. In order to improve the naturalness of dialogues, several data driven methods for language generation [Theune, 2003] have been proposed. The task of generating natural language text in turn consists of several sub-tasks such as: macro-planning, micro-planning, referring expression generation *etc.*. Given the recent advancements in the field of ASR and NLU (for example: continuous speech recognition and interpretation) and state-of-the-art in dialogue management using RL, language generation in dialogue systems has received significant interest. As a result of this several machine learning-based methods for language generation in dialogue systems (for example [Janarthanam & Lemon, 2009; Lemon, 2011]) have been proposed in recent past.

Chapter 3 formally introduces MDPs and gives on overview on methods for solving MDPs. In order to introduce sample efficient dialogue optimization techniques proposed in this manuscript, MDPs ought to be introduced first.

# Chapter 3

# Markov Decision Processes

In case of an SDS, the dialogue manager essentially performs the role of a decision maker. The goal of the (restaurant information) dialogue manager is to help the tourist (*i.e.*, to present information about restaurants of user's choice). In order to achieve this, dialogue act selection at every time step should be made by having the final goal in foresight. Let us assume that these decisions are made at time steps $t_1, t_2, ...t_n$, where $n$ is the number of turns required to accomplish the dialogue task[1]. At each time step $t_i$, the dialogue manager has to choose a suitable dialogue act given the dialogue state (which summarizes the dialogue history). Not all actions are allowed to be performed at every time step $t_i$. For instance, informing the list of restaurants before seeking the user preferences (*i.e.*, filling the location, price-range, restaurant-type related information) is deemed to be an inappropriate choice of action. However, it is likely that the dialogue manager will have more than one action to choose from at every time step $t_i$. Thus a dialogue policy plays a key role in spoken dialogue management. Given the dialogue state, a dialogue policy dictates what action should be performed next by the dialogue manager. The structure and the significance of a dialogue policy are discussed in Section 2.2.2. However, how to determine a good dialogue policy is a key question to be addressed.

In machine learning (ML), paradigms such as Markov Decision Process (MDP) [Bellman, 1957a; Puterman, 1994] and Partially Observable Markov Decision Pro-

[1] Note that the dialogue length ($n$) is subjective to user (behavior and goal) and thus tends to vary from one dialogue to another.

cess (POMDP) [Astrom, 1965] provide an efficient way to handle sequential decision making problems. This chapter focuses on casting the restaurant information dialogue manager as an MDP and describes how it can be solved in order to estimate a fairly good dialogue policy. MDPs can be optimized using Dynamic Programming [Bellman, 1957a] or Reinforcement Learning [Sutton & Barto, 1998]. In order to employ dynamic programming it is necessary to have the set of Markovian state transition probabilities and the associated rewards. However, this may not be always possible. For example, knowing the state transition probabilities for an MDP with finite but large state-action space is often not practical. Hence, reinforcement learning based policy optimization is commonly used in the dialogue domain. The organization of this chapter is as follows: Section 3.1 formally defines an MDP and Section 3.2 provides an overview on solving MDPs. Following which in Section 3.3 dynamic programming (model) based solutions are discussed. Section 3.4 outlines reinforcement learning (model-free) based policy optimization techniques. Finally in Section 3.5 statistical dialogue management using MDPs is discussed in detail.

## 3.1 Formal definition of an MDP

MDPs are a well known framework for modelling and solving sequential decision problems. Formally, an MDP is defined as a tuple $\{S, A, P, R, \gamma\}$, where $S$ is the (finite) state space, $A$ the (finite) action space, $P : S \times A \rightarrow \mathcal{P}(S)$ a set of Markovian transition probabilities. $R : S \times A \times S \rightarrow \mathbb{R}$ the reward function and $\gamma$ is the discount factor for weighting long-term rewards.

At each time step $t$, the agent (*i.e.*, the decision maker) in state $s_t \in S$ chooses and performs an action $a_t \in A$ (according to a policy $\pi$ being followed by the agent) and then steps to $s_{t+1}$ according to $p(.|s_t, a_t)$. In this case, $\mathcal{P}(S)$ determines with what probability ($p(s_{t+1}|s_t, a_t)$) state transition occurs from $s_t$ to $s_{t+1}$. The agent receives a reward or some feedback of the form $r_t = R(s_t, a_t, s_{t+1})$ (which basically measures the quality of the agent's choice of action). The agents goal is to maximize the cumulative reward that can be obtained over a period of time (but not necessarily maximize immediate rewards).

While using MDPs for modelling a problem under study, it is important that the Markovian assumption holds true. Markovian assumption states that: the transition of an agent from $s_t$ to $s_{t+1}$ depends only on the current state-action pair $(s_t, a_t)$ and not on the history of transitions (or the path followed to reach it). It is always possible (but not practical) to make a system Markovian by taking into account all the past transitions into the state representation. By using information state update paradigm for the dialogue state representation, the dialogue history is summarized implicitly in the dialogue state. This enables us to model dialogue problems as MDPs.

The *solution of an MDP* is called a *policy*, which in simple terms is a mapping from state to action: $\pi : S \rightarrow A$. The quality of such a policy is quantified by the so-called value function $V^\pi : S \rightarrow \mathbb{R}$, which associates each state $s \in S$ to the expected cumulative discounted reward that can be obtained by starting from state $s$ and from then on following the policy $\pi$:

$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) | s_0 = s, a_t = \pi(s_t), s_{t+1} = p(.|s_t, a_t)]. \tag{3.1}$$

The expectation term over trajectories in the definition of state-value function $(V^\pi(s))$ comes from the fact that trajectories are stochastic (due to the randomness of transitions). The optimal policy $\pi^*$ is the one for which the value $(V^\pi)$ is maximum for every state $s \in S$:

$$\pi^* \in \operatorname*{argmax}_{\pi} V^\pi. \tag{3.2}$$

The action-value function $Q^\pi : S \times A \rightarrow \mathbb{R}$ (defined as follows) can also be used to measure the quality of the associated policy $\pi$. The $Q$-function associates each state-action pair $(s_i, a_i)$ to the expected cumulative discounted reward that can be obtained by starting from $s_i$, performing action $a_i$ and from then on following the policy $\pi$:

$$Q^\pi(s, a) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) | s_0 = s, a_t = \pi(s_t), s_{t+1} = p(.|s_t, a_t)]. \tag{3.3}$$

In comparison to the value function, the action-value function *i.e.* the $Q$-function provides an additional degree of freedom concerning the choice of the first action to be taken. Actually, an important concept in RL is the greedy policy, which associates to a state the action which maximizes the expected cumulative discounted reward according to a currently estimated value function. Considering the value function, computing a greedy policy requires knowing the model, that is transition probabilities and the reward function. In the less constrained context, this model is not known neither learned. Thanks to this additional degree of freedom, a greedy policy can be computed from the $Q$-function without knowing the model. Based on the associated $Q^\pi$-function, the optimal policy $\pi^*$ is the one for which the value $(Q^\pi)$ is maximum for every state $(s, a) \in S \times A$:

$$\pi^* \in \operatorname*{argmax}_{\pi} Q^\pi \tag{3.4}$$

## 3.2 Solving MDPs

The most compelling aspect of modelling sequential decision making problems as MDPs comes from the fact that, the underlying mathematical framework governing MDPs provide an effective means for identifying and retrieving the optimal solution (*i.e.*, to find the optimal policy). Most existing solutions for solving MDPs revolve around using two fundamental equations; (i) Bellman evaluation equation and (ii) Bellman optimality equation [Bellman, 1957a]. This section provides an overview on how these equations can be used to determine the optimal solution for MDPs.

Recollect the definition of the state-value function $(V^\pi)$ in eq. 3.1. Computing the value function $V^\pi$ from its definition is indeed not a practical option. Such a solution would mean sampling trajectories in order to find $V^\pi$. However, there exist much effective means to determine $V^\pi$. Based on the Markovian assumption, it can be shown that the value function $V^\pi$ satisfies the *Bellman evaluation* *equation*:

$$\begin{aligned} V^\pi(s) &= E_{s'|s,\pi}[R(s,\pi(s),s') + \gamma V^\pi(s')] && (3.5) \\ &= \sum_{s' \in S} p(s'|s,\pi(s))\left(R(s,\pi(s),s') + \gamma V^\pi(s')\right) && (3.6) \end{aligned}$$

The operator $T^\pi$ associated with this equation is often termed as the Bellman evaluation operator. $T^\pi$ can be defined as:

$$T^\pi : V \in \mathbb{R}^S \to T^\pi V \in \mathbb{R}^S : [T^\pi V](s) = E_{s'|s}[R(s,\pi(s),s') + \gamma V(s')] \quad (3.7)$$

Bellman evaluation operator is indeed a contraction [Puterman, 1994] for which the optimal value function is the unique fixed-point:

$$V^\pi = T^\pi V^\pi \quad (3.8)$$

The other Bellman equation providing directly the optimal value function $V^*$ is the *Bellman optimality equation*:

$$\begin{aligned} V^*(s) &= \max_{a \in A} E_{s'|s,a}[R(s,a,s') + \gamma V^*(s')] && (3.9) \\ &= \max_{a \in A} \sum_{s' \in S} p(s'|s,a)(R(s,a,s') + \gamma V^*(s')) && (3.10) \end{aligned}$$

The optimal value function can be defined as the unique fixed-point of a similarly defined Bellman optimality operator $T^*$, which can also be shown to be a contraction [Puterman, 1994]. Thanks to the Banach theorem, for any initial value function $V_1$, the iterative scheme $V_{k+1} = T^* V_k$ converges to the optimal value function $V^*$. Thus using $V^*$ and the model, policy $\pi^*$ can be computed. However, it is more practical to work directly with the action-value function (*i.e.* the $Q$-function). The action-value function ($Q^\pi$) of a given policy $\pi$ and the optimal action-value function ($Q^*$) satisfies the Bellman evaluation and optimality equations respectively. Thus, as for as the action-value function concerned, $Q^\pi$ and $Q^*$ are the fixed-points of associated Bellman operators $T^\pi$ and $T^*$.

$$Q^{\pi}(s,a) = E_{s'|s,a}[R(s,a,s') + \gamma Q^{\pi}(s',\pi(s'))] \tag{3.11}$$

$$Q^{*}(s,a) = E_{s'|s,a}[R(s,a,s') + \gamma \max_{b \in A} Q^{*}(s',a')] \tag{3.12}$$

## 3.3 Dynamic programming

Dynamic programming [Bellman, 1957a] is a set of algorithms to solve problems which involve sequential decision making problems. If the sequential decision problem modelled as an MDP has a finite state space and is episodic (*i.e.*, a finite horizon problem), dynamic programming guarantees exact solutions. We know that the solution for an MDP is a policy and the optimization process aims at retrieving the optimal policy. Bellman evaluation and Bellman optimality equation forms the mathematical foundation for dynamic programming. Policy optimization using dynamic programming involves solving either of these equations. The optimization problem in itself is divided into several sub-problems and solved recursively to find the optimal solution. Computational results (such as $V(s_i)$) obtained during the optimization process are stored in the form of look-up table and re-used when needed.

It is important to note that two key assumptions ought to be met in order to employ dynamic programming for policy optimization: (i) the model of the system is well defined, *i.e.*, the Markovian transition probabilities ($p(s_{t+1}/s_t, a_t)$) and the reward function ($R(s_{t+1}, s_t, a_t)$) are made available (ii) the state space and the action space must be finite in size and the problem dealt with is episodic (finite horizon). However, there exists methods which can compute approximate solutions (from a set of transition samples) for problems with infinite state or action space (see Chapter 5). This section outlines two dynamic programming algorithms: (i) policy iteration - aims at estimating an optimal policy $\pi^*$ by iteratively and alternatively switching between policy evaluation and policy improvement and (ii) value iteration - aims at directly retrieving the optimal value function ($V^*$ or $Q^*$) so that the associated optimal policy $\pi^*$ can be reached.

In order to introduce policy iteration algorithm, policy evaluation and policy improvement ought to be outlined. Given two policies $\pi_1$ and $\pi_2$, it is difficult to directly determine the best one of the two. However, using the associated value functions $V_1$ and $V_2$ it is possible to conclude which of the two policies is better. Policy evaluation is the process of retrieving the value functions ($V^\pi$ or $Q^\pi$) associated to a policy $\pi$. Based on Bellman evaluation equation, $V^\pi$ can be defined as follows:

$$\begin{aligned} V^\pi(s) &= E_{s'|s,\pi}[R(s,\pi(s),s') + \gamma V^\pi(s')] \\ &= \sum_{a\in A}\pi(s,a)\sum_{s'\in S}p(s'|s,a)\left(R(s,a,s') + \gamma V^\pi(s')\right) \end{aligned} \tag{3.13}$$

As explained in 3.2, the existence of unique $V^\pi$ for $\pi$ is guaranteed and it is indeed the fixed point of the Bellman evaluation operator (refer eq 3.7). Thus, as the value of k $\rightarrow \infty$, $V^\pi_{k+1} \rightarrow V^\pi$ (see Eq. 3.8). This iterative process is summarized as shown in Eq. 3.14. Commonly used stopping criterion for policy evaluation is $\|V_{k+1} - V_k\| \leq \eta$, where $\eta$ is a small position constant.

$$\begin{aligned} V^\pi_{k+1}(s) &= E_{s'|s,\pi}[R(s,\pi(s),s') + \gamma V^\pi_k(s')] \\ &= \sum_{a\in A}\pi(s,a)\sum_{s'\in S}p(s'|s,a)\left(R(s,a,s') + \gamma V^\pi_k(s')\right) \end{aligned} \tag{3.14}$$

Given an arbitrary policy $\pi_k$, policy improvement aims at finding a policy $\pi_{k+1}$ (if it indeed exists) such that $V^{\pi_{k+1}} \geq V^{\pi_k}$. The improved policy $\pi_{k+1}$ is guaranteed by the policy improvement theorem, to be better if not as good as policy $\pi_k$. Using the computed value function $V^{\pi_k}$, the policy $\pi_k$ can be improved (during policy improvement) by being greedy with respect to $V^{\pi_k}$:

$$\pi_{k+1} = \pi_{\text{greedy}}(V^{\pi_k}) : s \longrightarrow \underset{a\in A}{\operatorname{argmax}}\, E_{s'|s,a}[R(s,a,s') + \gamma V^{\pi_k}(s')] \tag{3.15}$$

### 3.3.1 Policy iteration

Policy iteration converges to the optimal policy ($\pi^*$) by *iteratively* and *alternatively* performing policy evaluation and policy improvement. To begin with the value function ($V^{\pi_0}$) associated to some arbitrary policy $\pi_0$ is estimated (policy evaluation). Using the value function ($V^{\pi_0}$), policy $\pi_0$ is improved resulting in a better policy $\pi_1$ (policy improvement). Policy $\pi_1$ is evaluated to determine $V^{\pi_1}$, which is then used to improve the policy and so on. Policy iteration can be summarized as shown below:

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} ... \pi_{k-1} \rightarrow V^{\pi_{k-1}} \rightarrow \pi_k \rightarrow V^{\pi_k} \tag{3.16}$$

It can be shown that under this scheme, $V^{\pi_{k+1}} \geq V^{\pi_k}$ [Puterman, 1994]. If this equality holds true, the optimal policy can be reached (the Bellman optimality equation being satisfied in this case). Since the number of different policies is finite (as a result of finite state space) the optimal policy can be reached in a finite number of iterations. Being an iterative algorithm, it must be terminated at some point. Stopping criterion can be defined such that policy iteration terminates if the number changes made to the policy is zero or negligibly small. This algorithm can be extended for the action-value function in which case the greedy policy is defined as follows:

$$\pi_{k+1} = \pi_{\text{greedy}}(Q^{\pi_k}) : s \rightarrow \operatorname*{argmax}_{a \in A} Q^{\pi_k}(s, a) \tag{3.17}$$

### 3.3.2 Value iteration

Value iteration aims at determining the optimal policy ($\pi^*$). However, it primarily focuses on retrieving the optimal value function $V^*$. The optimal value function $V^*$ is actually the value function associated with optimal policies $\pi^*$. Thus by using $V^*$ and the model it is possible to retrieve the optimal policy by simply acting greedily with respect to $V^*$.

$$\pi^* = \pi_{\text{greedy}}(V^*) : s \rightarrow \operatorname*{argmax}_{a \in A} E_{s'|s,a}[R(s, a, s') + \gamma V^*(s')] \tag{3.18}$$

It may be useful to recollect that $V^*$ is indeed the unique fixed-point of the contraction $T^*$. Thanks to the Banach theorem, for any initial value function $V_1$, the iterative scheme $V_{k+1} = T^* V_k$ converges to the optimal value function $V^*$ [Puterman, 1994]. Since the convergence is asymptotic (*i.e.*, $V_k \rightarrow V^*$ as

$k \to \infty$.) with regard to value iteration, a stopping criteria must be introduced. Commonly used stopping criterion is $\|V_{k+1} - V_k\| \leq \eta$, where $\eta$ is a small positive constant. Value iteration can be summarized as show here:

$$\forall s \in S, \quad V_{k+1}(s) = \max_{a \in A} E_{s'|s,a}[R(s,a,s') + \gamma V_k(s')] \tag{3.19}$$

$$V_0 \to V_1 \to V_2 \to ...V_{k-1} \to V_k \tag{3.20}$$

## 3.4 Reinforcement learning

When the model of the problem is not available, reinforcement learning [Sutton & Barto, 1998] can be employed for policy optimization. The reinforcement learning agent (decision maker) interacts with the environment as shown in Figure 3.1. The agent maintains an internal state $s$, which is the view of the environment from its perspective. At any given time $t$, based on the current state $s_t$, the agent chooses and performs an action $a_t$. Using its sensory inputs (used to observe the changes in the environment) the agent steps from the current state $s_t$ to the next state $s_{t+1}$. The environment provides a feedback $r_t$ to the agent. This feedback $r_t$ quantifies the quality of the agent's decision to perform action $a_t$ from the state $s_t$.

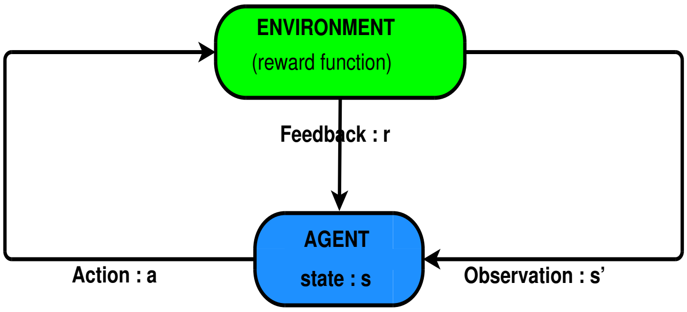


Figure 3.1: Reinforcement Learning: Agent interacting with the environment

The reward function succinctly represents the task to be accomplished by the agent. In order to use reinforcement learning it is necessary that this reward function is known beforehand. The agent behavior in itself can be modelled as an MDP. The state space of the MDP is defined by all possible internal states of the agent and the action space of the MDP includes all actions that can be performed by the agent. The Markovian transition probabilities need not be defined (since the policy optimization is performed in a model free set-up). The reinforcement learning agent must learn to optimize its behavior with regard to the reward function of the environment. Reinforcement learning based solutions for MDPs also rely heavily on the Bellman equations (eq: 3.7,3.10). To begin with, this section outlines the temporal-difference-based learning technique. Following which two RL algorithms are discussed: (i) $Q$-learning based on value iteration and (ii) SARSA based on policy iteration.

### 3.4.1 Temporal difference learning

Temporal differences (TD) is a prediction algorithm used for determining the value function $V^\pi$ associated to some arbitrary policy $\pi$. It is one of the most widely used learning scheme in reinforcement learning. At any given time step, temporal-difference-based learning methods use the future value of a variable to be predicted to update its current value:

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s)) \tag{3.21}$$

where $\gamma$ is the discount factor which determines to what extent $V(s)$ depends on future values such as $V(s'), V(s''), \ldots$ *etc.* Here, $\alpha$ is the learning rate which indicates the level of importance given for new estimate of $V(s) : (r + \gamma V(s'))$ in comparison to the existing estimate of $V(s)$. Setting $\alpha$ to 1, resets the value of $V(s)$ to $(r + \gamma V(s'))$, whereas $\alpha < 1$ allows prediction of $V(s)$. The following algorithm from [Sutton & Barto, 1998] outlines TD updates for determining the value function $V^\pi$ associated to some policy $\pi$:

Unlike dynamic programming, TD uses partial backups. TD($\lambda$) determines the extent of backup to be provided. The algorithm presented here is the simplest case where $V(s)$ only depends on the immediate next state $V(s')$. TD can be extended to perform value iteration ($Q$-learning) and policy iteration (SARSA). Also it is important to note that TD algorithms are asynchronous (*i.e.*, learn

**Algorithm 1:** Temporal differences TD(0) based $V^\pi$ prediction

```
Given
$\pi$ some policy to be evaluated
Initialization
Initialize V(s) to some arbitrary value
Computation
for every episode do
    $s \leftarrow s_0$
    for every transition do
        $a \leftarrow \pi(s)$
        Perform: a Observe: r & s'
        $V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$
        $s \leftarrow s'$
        if s is terminal state then
            Break
        end
    end
end
```

after observing every sample).

### 3.4.2 SARSA

TD learning outlined here for policy evaluation can be extended to determine the action-value function. SARSA is a well-known online/on-policy RL algorithm. Here the policy being evaluated and the policy being used for data generation are one and the same. SARSA is the acronym for state-action-reward-state-action $(s, a, r, s', a')$. It is an optimistic policy iteration algorithm[1] which focuses on retrieving the optimal policy by iteratively alternating between policy evaluation and policy improvement. SARSA algorithm as described in [Sutton & Barto, 1998] is presented in Algo. 2.

SARSA employs asynchronous value iteration scheme, where upon observing a sample $(s, a, r, s', a')$ policy evaluation updates only the value estimate of $Q(s, a)$. This (partial) estimate of the value function is then used to perform

[1] Optimistic policy iteration scheme performs policy improvement without waiting for the policy evaluation step to return the (true) value function associated to the current policy.

**Algorithm 2:** SARSA: On-policy TD algorithm

```
.
Initialization
Initialize $Q(s,a)$ with arbitrary values
Computation
for every episode do
    $s \leftarrow s_0$
    for every transition do
        $a \leftarrow \pi(s)$ derived from $Q(s,a)$ using $\epsilon$-greedy
        Perform: a Observe: r & s'
        $a' \leftarrow \pi(s')$ derived from $Q(s',a')$ using $\epsilon$-greedy
        $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$
        $s \leftarrow s'$ and $a \leftarrow a'$
        if s is terminal state then
            break
        end
    end
end
```

policy improvement. Being an online RL algorithm SARSA suffers from the so-called exploration versus exploitation dilemma. Ways to cope with this dilemma is discussed in detail in Section 3.4.4

### 3.4.3 Q-Learning

$Q$-learning is a value iteration algorithm which aims at retrieving the optimal action-value function. It is an off-policy algorithm and the learning can be performed either in online or off-line setting and in a controlled manner. The $Q$-learning algorithm [Watkins & Dayan, 1992] is presented in Algo. 3.

Policy $\pi$ used for choosing $a$ given $s$, can be estimated by acting greedily with respect to $Q$-function or by combing it with an $\epsilon$-greedy action selection. Based on the principles of value iteration (recall Section 3.2), the learning outcome is expected to be the optimal value function and thus an optimal policy can be estimated upon termination. Being an off-policy algorithm, $Q$-learning can be used in a off-line setting for policy optimization using samples generated from a sub-optimal policy. However, when used in a online setting, $Q$-learning also

**Algorithm 3:** Q-Learning : Off-policy TD algorithm

```
Initialization
Initialize $Q(s, a)$ with arbitrary values
Computation
for every episode do
    $s \leftarrow s_0$
    for every transition do
        $a \leftarrow \pi(s)$ derived from $Q(s, a)$ using $\epsilon$-greedy
        Perform: a Observe: r & s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
        $s \leftarrow s'$ and $a \leftarrow a'$
        if s is terminal state then
            break
        end
    end
end
```

suffers from the exploration versus exploitation dilemma. How to cope with this dilemma is discussed in detail in Section 3.4.4

### 3.4.4 Taxonomy of RL algorithms

There exits several different taxonomies of RL algorithms. Most often these algorithms are classified as follows: (i) online and off-line, (ii) on-policy and off-policy, (iii) controlled and uncontrolled. The following section explains how these classifications can be made using different criterion. We know that in case of RL, policy optimization is performed using the observed samples either in the form of $(s, a, r, s')$ or $(s, a, r, s', a')$. Based on how these samples are generated, RL algorithms can be classified into two types: (i) online and (ii) off-line. Online algorithms obtain these samples from live interaction with the environment. Off-line algorithms perform policy optimization using samples which are already available. Based on the policy used for data generation and the policy being evaluated RL algorithms can be classified into two types: (i) on-policy and (ii) off-policy. In case of on-policy algorithms the policy being evaluated and the (control) policy being used for (action selection in order to) interaction with the environment are one and the same. Whereas, in case of off-policy algorithms the policy being evaluated and the policy being used for data generation are indeed different. Based on the policy used for data generation RL algorithms can be further classified into two types: (i) “controlled” and (ii) “uncontrolled”. In both these cases the interaction is still controlled. However, this classification addresses how the interaction is controlled (differentiates the control mechanism in place). In case of controlled RL algorithms the policy being used for data generation (control) depends on the value function being estimated. Whereas, if the policy used for data generation is not related to or not estimated using the value function being estimated then the RL algorithm is said to function in an uncontrolled manner.

Off-line RL algorithms must aim at optimizing an optimal policy from the available data (samples) and there exists no possibility to generate more samples (by means of interaction with the environment) during the process of optimization. Online and controlled algorithms (irrespective of on-policy or off-policy) suffer from exploration versus exploitation dilemma. While interacting the environment the RL agent should decide whether to explore or exploit from the experience it has gained from the past interactions. For such algorithms to be effective (for policy optimization) it is important to choose a good trade-off between exploration and exploitation. One simple possibility to cope with this challenge is to use an $\epsilon$-greedy action selection scheme. In this case action selection is performed in a greedy manner with probability (1 - $\epsilon$) and random action is selected with probability $\epsilon$. Choosing a (constant) value for $\epsilon$ close to 1 (but not equal to one) will result in moderate exploration and a good degree of exploitation. However, during the initial stages of learning, knowledge available for the RL agent (in the form of value function estimates) is limited and hence it is ideal to give preference for exploration. However, as the learning progresses, the RL agent tends to have a good estimate of the value function and thus it is ideal to give preference for exploitation. A simple trade-off for handling exploration versus exploitation dilemma is to involve a decaying exploration rate (rather than a constant value for $\epsilon$). When the learning begins this rate can be high (so that the agent explores more) and the exploration rate can be halved or decayed as the learning progresses (so that the agent exploits more). It is also possible to employ efficient schemes such as bonus-greedy (explained in Chapter 6) which makes use of uncertainty related to the value estimates.

## 3.5 Dialogue management using MDP

So far, this chapter focused on solving sequential decision making problems modelled as MDPs using dynamic programming and reinforcement learning algorithms. The rest of this chapter will focus on building statistical dialogue management engines. We know that the dialogue management policy navigates the dialogue system to perform the specified task. However, dialogue policy estimation is a complex procedure given the variations in user behavior and uncertainty involved due to speech recognition and language understanding errors.

### 3.5.1 Casting dialogue management problem as an MDP

If the dialogue management task is casted as an MDP, then the agent behavior can be optimized using dynamic programming or reinforcement learning. Let us term the MDP-based dialogue manager as *MDP-SDS*. We know that an MDP is formally defined as a tuple $\{S, A, P, R, \gamma\}$. Thus, the first step in casting the dialogue management task as an MDP is to define the state space $S$, action space $A$ and reward function $R$ of MDP-SDS. Let us revisit the restaurant information dialogue system outlined in Section 2.2. The dialogue state of MDP-SDS takes the form: $\{ \ \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6 \ \}$. Recall the list of actions (action space of the MDP-SDS) specified in Section 2.2.2: ask-destination-type, ask-restaurant-type, ask-restaurant-location, ask-restaurant-price-range, inform-restaurants, greet-user and close-dialogue.

We know that $P$ represents the set of all Markovian transition probabilities. In order to compute this statistics from a corpus, every dialogue state has to be visited, multiple number of times. Also it depends on the corpus to showcase all possible state transitions. Thus, it is impractical to compute these probabilities from the corpus. For this reason, it is more convincing to employ model free policy optimization using reinforcement learning.

Having defined the state space and the action space of MDP-SDS, the next important task is to define the reward function. It may be useful to recall that the RL-based agent behavior optimization maximizes the agent behavior with regard to the specified reward function. Thus reward function specification is a critical task in casting dialogue management problem as an MDP. It succinctly represents the dialogue task that has to be achieved by the dialogue manager. Yet, there is no straightforward means for defining the reward function. Often reward functions are defined by the system designer based on their domain expertise.

The optimal dialogue strategy is expected to maximize the reward that can be obtained by the dialogue manager. This implicitly guarantees user satisfaction and dialogue task accomplishment [Singh *et al.*, 1999; Walker *et al.*, 1997]. Since the exact reward function cannot be observed from data, some subjective reward function is estimated from a linear combination of objective measures (such as: dialogue task completion, dialogue length, number of speech recognition errors, *etc.*). It is customary to specify a reward function [Levin & Pieraccini, 1998], perform dialogue optimization and observe the learned or optimized behavior. If the optimized behavior is not in correlation with the expected behavior, the reward function is fine tuned and the entire process is repeated.

- turn penalty : -1
- task-completion reward : 100

*Restaurant information SDS: example reward function*

Let us analyse the example reward function for the restaurant information MDP-SDS. This reward function is a linear combination of measures such as: turn penalty, task completion. Turn penalty penalizes agent performance based on the length of the dialogue episode. A dialogue task is deemed to be complete if the intended purpose of the dialogue is achieved before the end of the dialogue episode. For example, in case of the restaurant information dialogue system, the dialogue task (to furnish user requested information) is said to be completed if the terminal dialogue state is $\{1, 1, 1, 1, 1, 1\}$. Task completion reward implicitly underlines the importance of task completion and thus ensures the dialogue manager to accomplish the specified task.

### 3.5.2 Dialogue policy optimization using RL

As discussed earlier RL based optimization can be performed under two different settings: (i) online RL algorithms or (ii) off-line (off-policy) algorithms. We know that in case of RL the actual learning is performed by observing $(s, a, r, s', a')$ samples. A RL policy learner can observe these samples either from its interaction with users or perform learning from samples observed in the dialogue corpus (learn from existing data). If a dialogue corpus is available, off-line (*i.e.*, off-policy) RL algorithms such as $Q$-learning can be used to learn the underlying optimal policy. Once the initial policy is learned, (if necessary) it can be further improved under an online setting. Online/on-policy algorithms tend to directly update the policy being estimated. These changes at times may result in unexpected fluctuations in the system behavior. Since the policy being estimated is also used for data generation, these policy fluctuations are often visible to users. This presents inconvenience for users and often result in unsuccessful dialogues (for instance, users may hangup). However, online/off-policy algorithms don't suffer from this shortcomings (since control policy is different from policy being optimized).

Also involving real users for the sake of dialogue corpora generation or online policy optimization is an expensive and time consuming process. However, commonly used reinforcement learning algorithms such as SARSA and $Q$-learning are sample inefficient and need large amounts of data. In order to cope with this data requirement problem user simulators were introduced. Existing approaches for building user simulations and their role in dialogue optimization are discussed in detail in Section 4.1. *To summarize, an ideal algorithm for dialogue optimization must have the ability to learn in a off-line setting (learn an initial policy from fixed amount of dialogue corpora) and then continue learning in a online/off-policy setting (improve the initial policy by interacting with real users). Irrespective of the algorithm employed for policy optimization it should be sample efficient (so that the necessity for user simulation can be eliminated).*

### 3.5.3 Dialogue policy evaluation schemes

Dialogue policy evaluation[1] schemes are intended to quantify the quality of the dialogue policies. Dialogue optimization yields dialogue policies which are meant to be optimal. However, most of the algorithms (example: policy iteration) doesn't guarantee the effectiveness of the retrieved dialogue policies. Thus it is essential to evaluate dialogue policies before it can be used in real-life scenarios. The most effective way of evaluating dialogue policies is to use them and interact with users. However, employing real users to evaluate several dialogue policies is a time consuming procedure.

One simple yet effective way to overcome the difficulties of policy evaluation using human users is to employ user simulation for evaluation. In this case, dialogue policies are evaluated based on the reward function used during dialogue optimization. The cumulative reward function quantifies the performance of a dialogue policy. Performance observation measures can be computed from a set of dialogues generated from the interaction between the dialogue manager and user simulation. The performance of the dialogue policies can then be quantified based on the average reward obtained by them. Objective measures computed from a set of dialogues can also be used for policy evaluation: average success rate (defined as the percentage of dialogues with successful task completion in comparison to the total number of dialogues), average dialogue length, robustness to speech recognition errors, *etc.*

[1] In reinforcement learning, policy evaluation often corresponds to retrieving the value function ($V^{\pi}$) associated with some policy ($\pi$). However, in case of the dialogue domain, it refers to the process of measuring the effectiveness of a policy in accomplishing a dialogue task.

# Chapter 4

# User simulation in dialogue systems

User simulators [Schatzmann *et al.*, 2006b] play a central role in designing statistical SDS. It is used to generate synthetic dialogues based on some reference dialogue corpus. These dialogues can then be used for dialogue optimization and policy evaluation. The organization of this chapter is as follows: Section 4.1 introduces the role of user simulation in SDS, following which an overview on some of the existing approaches for building user simulators is presented. Section 4.2 lists commonly used metrics for evaluating the performance of user simulators. Section 4.3 outlines few shortcomings in present day user simulation techniques.

## 4.1 User simulation: an overview

User simulators are computational models which exhibit behavior similar to that of human users. Its primary goal is to generate an appropriate user utterance for any given dialogue act. User behavior simulation can be performed at three different levels: (i) *signal-level*, (ii) *word-level* and (iii) *intention-level*. Signal-level simulation discussed in [Götze *et al.*, 2010] focuses on generating user utterances in the form of speech signals. Word-level simulation as discussed in [Jung *et al.*, 2009] aims at generating text based user utterances. However, in order to perform dialogue optimization most often simulation of user behavior is performed at the intention-level [Eckert *et al.*, 1997a].

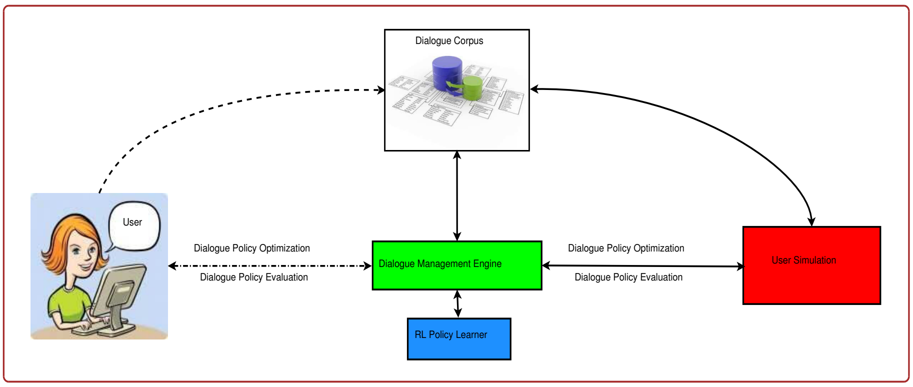

Dialogue Corpus
User
Dialogue Policy Optimization
Dialogue Policy Evaluation
Dialogue Management Engine
Dialogue Policy Optimization
Dialogue Policy Evaluation
User Simulation
RL Policy Learner

Figure 4.1: Dialogue Management using User Simulation

Dialogue management using user simulation is summarized in Figure 4.1. Dialogue optimization using RL can be performed in two methods: (i) online (on-policy or off-policy) learning based on interaction, (ii) batch methods that optimize a policy directly from dialogue corpora. In both cases user simulators can be employed for dialogue policy optimization. In case of interaction based learning; RL based policy learner (of the dialogue manager) interacts with the user simulation. Here user simulation responds to the dialogue acts with suitable user acts. In case of the batch methods (discussed in detail in Section 5) synthetic dialogue corpora generated using user simulation is used for policy optimization. Here the intention is to make use of the available dialogue corpora to learn a fairly good initial policy for dialogue optimization. Such an initial policy can then be improved by continued learning in an online set-up.

As illustrated in Figure 4.1, user simulators are often built using existing dialogue corpus. Data driven methods (such as [Eckert *et al.*, 1997a; Georgila *et al.*, 2005; Pietquin & Dutoit, 2006; Schatzmann *et al.*, 2007a]) for user modelling have evolved to be the state-of-the-art. These methods focus on building computational models based on some reference dialogue corpus. To begin with, dialogue corpus is annotated in terms of dialogue acts and user acts. On a general case: annotated dialogue corpus is used to learn conditional probability distributions, which are then used to choose (the most probable) an user act given the dialogue act.

There exists several possibilities for computing these conditional probability distributions from dialogue corpus. Existing methods for user simulation focuses on defining and computing slightly different variants of this conditional probability distributions. An overview on some of these approaches is presented in the following sections.

### 4.1.1 Probabilistic user simulation

Transition-level user simulation (for example [Eckert *et al.*, 1997a]) focuses on reproducing the observed user behavior at the level of dialogue transitions. A simple approach for such simulation is to use the frequency of different user acts given a specific dialogue act. Conditional probability distributions for user acts given dialogue acts are computed from the reference dialogue corpus. Synthetic dialogues generated using this method tend to correlate with the reference dialogue corpus at the transition level (*i.e.*, statistically coherent in terms of frequency of user act occurrence). However, dialogues generated using transition-level simulation often lag in terms of overall consistency.

$$u_t = \underset{i}{\operatorname{argmax}}\ p(u_i|d_t) \tag{4.1}$$

where $i = 1..n$ represents the number of all possible user acts. For instance, let us say that the dialogue act ($d_t$) is observed by the user simulation at turn $t$. Here the choice of user act ($u_t$) is made based on the equation 4.1. The user act which has the highest probability (of all possible user acts) is chosen by the user simulation.

### 4.1.2 $n$-gram user simulation

A probabilistic user simulation termed as n-gram user was proposed in [Georgila *et al.*, 2005]. It aims at reproducing partial-trajectories (sequence of user acts) as observed in the dialogue corpus. Here $n$ corresponds to the length of the dialogue history to be considered while choosing an user act. Apart from conditioning the probability of user act on the current dialogue act, $n$ prior user acts from the history of dialogue episode are also considered.

$$u_t = \underset{i}{\text{argmax}}\ p(u_i|d_t, u_{(t-1)}, u_{(t-2)}...u_{(t-n)}) \tag{4.2}$$

User act selection by n-gram user is summarized in equation 4.2. The conditional probability of current user act($u_t$), current dialogue act ($d_t$) and the $n$ previous user acts $u_{t-1}, u_{t-2}...u_{t-n}$. This results in a user act selection methods which is based on the occurrence frequency of a sequence of user acts (as observed in some reference dialogue corpus). Overall consistency of synthetic dialogues is expected to be better when compared to that of data from transition-level simulation.

### 4.1.3 Bayesian Networks based user simulation

Dynamic Bayesian Networks (DBN) [Pietquin *et al.*, 2009] based user simulation aims at casting the user simulation problem as a probabilistic graphical model (as shown in Figure 4.2). Similar to n-gram user, DBN user simulator focuses on reproducing sequence of user acts (partial-trajectory-level simulation). Being a generative model DBN simulator can be employed to predict the user act. If some reference dialogue corpus is available, DBN parameters can be learned from the corpus. However, it is also possible to hand-craft these parameters to build a hand-crafted user simulation.

Interaction between the user and the dialogue manager is considered as a sequential transfer of intentions organised in the form of dialogue turns. User act prediction using the Bayesian user simulation can be summarised as shown in equation 4.3. It chooses an user act $u_t$, based on the current dialogue act ($d_t$), user's goal ($g_t$) and knowledge ($k_t$). Here, $k_t$ represents what information has been exchanged until reaching turn $t$ and $g_t$ represent what is yet to be achieved. After each dialogue turn user goal and user knowledge are updated.

$$u_t = \underset{i}{\text{argmax}}\ p(u_i|d_t, g_t, k_t) \tag{4.3}$$

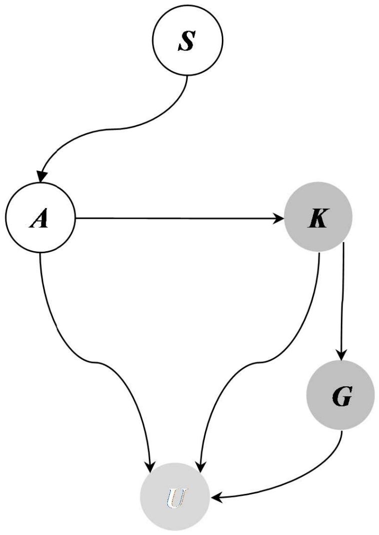


Figure 4.2: Bayesian Network-based User Simulation

### 4.1.4 Advanced n-gram user simulation

Advanced n-gram user simulation proposed by [Georgila *et al.*, 2006] is a variant of n-gram user simulation (see Section 4.1.2). In this case information such as whether a specific slot has been filled or not is also used while computing the conditional probability of an user act. This inclusion provides an implicit representation of the goal of user simulation. Advanced n-gram user simulation for a two slot $(s1, s2)$ dialogue problem is summarized as:

$$u_t = \underset{i}{\text{argmax}} \ p(u_i|d_t, u_{(t-1)}...u_{(t-n)}, s1_{(filled)}, s2_{(empty)}) \tag{4.4}$$

### 4.1.5 Agenda based user simulation

Agenda based user simulation proposed by [Schatzmann *et al.*, 2007b] maintains an internal representation of the user goal and agenda (which are used during user act selection process). The agenda $(a_t)$ of user simulation in itself can be visualized as a list of user acts or intentions to be performed in order to achieve the goal. The goal $(g_t)$ can be generated using a set of structured rules such as ontologies. The goal of the user simulation is then broken down into user acts, which are then stacked to form the agenda of the user simulation. User act selection in case of agenda based user simulation can be summarized as:

$$u_t = \underset{i}{\operatorname{argmax}}\ p(u_i|d_t, g_t, a_t) \tag{4.5}$$

Since real users are goal directed, our focus during user modelling should be on simulating dialogue trajectories. However, the primary challenge in such an approach is to determine the goal of the user. Once the goal of the user simulation is determined it can be used in several different forms (example: goal, agenda, etc). Secondly, even if the goal information is available, the user behavior must be subjective with regard to the dialogue context (similar to the ability of real users). This cannot be achieved by (only) conditioning the selection of user act on the dialogue act and user goal. A possible solution to cope with these challenges is to present rich information about the dialogue context, the current dialogue act and the user goal in the form of an information state (similar to dialogue management). Unlike existing methods, Inverse Reinforcement Learning (IRL) based user simulation provides a more formalized approach to retrieve the user goal. In this case, some reward function which can explain the real user behavior observed in the dialogue corpus is predicted using IRL. This function is used to optimize the simulated user (modelled as an MDP) behavior. More information on IRL user simulation can be found in Chapter 7.

## 4.2 User modelling evaluation metrics

User simulators, when used for dialogue optimization, tend to have a direct impact on the quality of the retried dialogue policy as shown by [Schatzmann *et al.*, 2005]. Thus, it is necessary to ensure good performance of user simulators. However, there exists no commonly accepted list of evaluation metrics. Evaluation with human intervention would be a possibility, nevertheless user simulators are introduced in first place to reduce human involvement. For this reason, several automatic schemes have been proposed to measure the performance evaluation. As suggested in [Pietquin & Hastie, 2011], evaluation measures are grouped under two schemes: (i) turn-level evaluation measures (outlined in this section) and

(ii) dialogue-level measures. Under both these schemes, the performance of user simulations are measured based on the quality of synthetic dialogues generated by them. Some definitions and notations used in this chapter were first presented in [Pietquin & Hastie, 2011].

To begin with most common, turn-level evaluation metrics are presented in this section. They aim at evaluating the performance of user simulation at the level of dialogue transitions. Trivial information such as the prediction capability of user simulators are measured using these metrics. For example, set of synthetic dialogues generated using user simulation are expected to correlate with the reference dialogue corpus in terms of action frequency. However, it is important to note that turn level metrics focus on evaluating the local consistency and not the global consistency of the synthetic dialogues.

### 4.2.1 Precision and Recall

Precision and Recall are common measures in machine learning to measure a model's ability to predict the observed values. Since user simulators are used to predict user acts (see [Zukerman & Albrecht, 2001]), their performance can be measured using precision and recall metrics. These metrics can be adapted for user modelling as shown in [Schatzmann *et al.*, 2006b]. They are some of the widely used evaluation metrics in dialogue domain. Precision and Recall are defined as follows:

$$\text{Precision:}\quad P \quad = \quad 100 \times \frac{\textit{Correctly predicted actions}}{\textit{All actions in simulated response}} \tag{4.6}$$

$$\text{Recall:}\quad R \quad = \quad 100 \times \frac{\textit{Correctly predicted actions}}{\textit{All actions in real response}} \tag{4.7}$$

These two measures are complementary and cannot be used individually to rank user simulation methods. However, the balanced $F$-measure [van Rijsbergen, 1979] can be used to combine these measures into a single scalar:

$$F = \frac{2PR}{P+R} \tag{4.8}$$

Precision and recall metrics, do not measure the generalisation capabilities of the user simulators. As a matter of fact, it penalizes attempts to generalise when the model generates unseen dialogues.

### 4.2.2 Kullback-Leibler (KL) divergence and dissimilarity

The Kullback-Leibler (KL) divergence [Kullback & Leibler, 1951] is a measure of dissimilarity between two probability distributions P and Q. KL divergence is defined as follows:

$$D_{KL}(P||Q) = \sum_{i=1}^{M} p_i log(\frac{p_i}{q_i}) \tag{4.9}$$

KL divergence was first used as an user simulation evaluation metric in [Cuayáhuitl *et al.*, 2005]. While using KL divergence for evaluating user simulations, $P$ represents the probability distribution of user acts in some reference dialogue corpus while $Q$ is the distribution of user acts in the synthetic dialogues. However, since the KL divergence is not a distance (since it is not symmetric), the dissimilarity metric $DS(P||Q)$ is introduced:

$$DS(P||Q) = \frac{D_{KL}(P||Q) + D_{KL}(Q||P)}{2} \tag{4.10}$$

Since KL divergence is a measure used to predict dissimilarities between two distributions, it provides a natural means for evaluating user simulations. However, since it is an unbounded metric it cannot be used directly for ranking user simulations.

### 4.2.3 Log-likelihood

Dialogue-level metrics for evaluating user simulation aim at using the entire dialogue trajectories. The simulated behavior observed in the synthetic dialogues (using dialogue-level metrics) are compared to that of dialogues present in some reference dialogue corpus. Considering entire dialogue trajectories during evaluation provide an implicit measure of overall dialogue consistency. Rest of this section outlines the most commonly used dialogue-level evaluation metrics.

The log-likelihood $\mathcal{L}(x)$ of a data set $x = \{x_i\}_{i=1,\ldots,N}$ given a model $m$, is defined as $\mathcal{L}(x) = \log p(x|m) = \log p_m(x)$. If the data samples $x_i$ are assumed to be independent (a common assumption), $\mathcal{L}(x)$ can be written as:

$$\mathcal{L}(x) = \log \prod_{i=1}^{N} p_m(x_i) = \sum_{i=1}^{N} \log p_m(x_i) \tag{4.11}$$

In case of user simulations, the higher the log-likelihood, the higher the consistency between the dialogue corpus and the user simulation being evaluated. Log-likelihood measures the user simulator's ability to predict sequences of user acts as observed in the reference corpus. The outcome of the this metric is positive and scalar (which can help to rank user simulators).

### 4.2.4 Bilingual Evaluation Understudy

The BLEU (Bilingual Evaluation Understudy) score [Papineni *et al.*, 2002] is often used in machine translation domain. It compares two semantically equivalent sentences by computing the geometric mean of the n-gram precisions with a brevity penalty to compensate for high n-gram precision of short utterances. In case of user simulation evaluation, the n-grams considered are not sequences of words but sequences of intentions. The later is termed as Discourse-BLEU (D-BLEU) [Jung *et al.*, 2009].

The BLEU score is known to be highly correlated with human judgement [Doddington, 2002; Papineni *et al.*, 2002], thus provides a compelling case for its use in evaluating user simulations. However, BLEU has been reported to fail to predict machine translation improvements and naturalness [Lee & Przybocki, 2005]. Since BLEU is a measure of semantic equivalence of (sequence of) intentions, any act of generalisation by user simulation is penalized.

### 4.2.5 Simulated User Pragmatic Error Rate

SUPER (Simulated User Pragmatic Error Rate) [Rieser, 2008; Rieser & Lemon, 2006] is a metric resulting from an effort to combine different metrics. Similar to computing word error rate for speech recognition systems in terms of insertions, deletions and substitutions, this metric combines scores for consistency ($I_s$ - in terms of insertions of user intentions), completeness ($D_s$ - in terms of deletions of user intentions) and variety ($V_s$ - comparing probabilities of human and user simulator intentions). For each scenario $s$, these individual scores $I_s$, $D_s$ and $V_s$ are computed and combined into a single score as follows:

$$\text{SUPER} = \frac{1}{m} \sum_{s=1}^{m} \frac{V_s + I_s + D_s}{n} \tag{4.12}$$

where $n$ is the number of possible user intentions and $m$ the number of contexts or scenarios. Since the resulting score is a scalar value it can be used to compare user simulators.

### 4.2.6 Performance of dialogue policy

User simulators are used in the first place to optimize dialogue policies. Thus an indirect method for evaluating an user simulation is indeed to evaluate the dialogue policy (learnt using the simulation being evaluated) with the help of real users. During the evaluation process, based on their interactions with the dialogue system, users provide feedback on the performance of dialogue policies. This feedback can be used to rank user simulations which were used to learn the dialogue policies. Evaluation of user simulation based on the performance of the dialogue policy is summarized in Figure 4.3

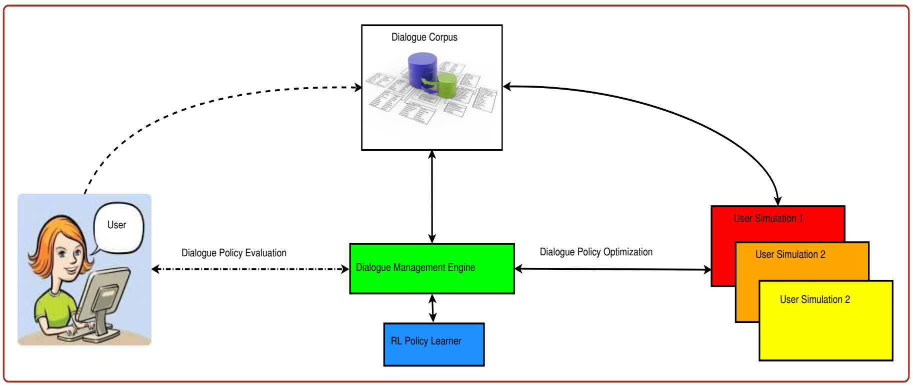


Figure 4.3: User simulation evaluation based on performance of dialogue policy

## 4.3 Revisiting user simulations

Almost all existing works on user simulation focus on generating synthetic dialogues which are statistically consistent with the reference dialogue corpus. A wide variety of evaluation metrics were introduced to ensure that this criteria for statistical coherence is met for a user simulation to be deemed good or bad. Since most methods for building user simulators are data driven, it is in fact logical to ensure whether the computational model predicts as expected.

However, there are two key problems: Firstly, too much of onus is left on the reference dialogue corpus generation. Yet, there is no guarantee for any dialogue corpus to be comprehensive. Often behaviors observed in the dialogue corpus can be attributed to a limited group or class of users. Since the dialogue corpus only presents certain aspects of the user behavior, focusing merely on reproducing it perhaps is not realistic means user modelling. This in fact results in some generic user behavior which happens to be an amalgamation of all user behaviors observed in the dialogue corpora. Often such generic user behaviors may indeed be non-existent user behaviors. Since the dialogue policy adapts to simulated user behavior it is important to present all different possible user behaviors rather than one generic behavior. Secondly, present day user simulators lag on aspects such as adaptation, generalization and evolution. Yet, these aspects are part and parcel of real user behavior. For example, adaptation to the other party involved in communication, generalisation and evolution during the progress of a dialogue are often exhibited in human-human interaction.

Today, there is a pressing necessity for developing SDS which can facilitate man-machine dialogues. Naturalness of man-machine dialogues is a key aspect that determines the wide acceptance of dialogue systems. However, effectiveness of dialogue policies can be attributed towards the quality of user simulators. Thus, in order to improve the performance of dialogue systems, focus should be on improving the performance of user simulators. In order to build a better user model, it is necessary to revisit the task of user simulation. Perhaps how this modelling problem is perceived or what is expected as the outcome ought to be changed. For instance, rather than focusing on just reproducing the statistical consistency our focus can be on building models that can adapt, generalize and in fact evolve (just like real users).

# Part II

# Sample Efficient Dialogue Optimization

# Chapter 5

# Approximate Dynamic Programming

It may be useful to recall that the methods for solving MDPs (discussed in Chapter 4) assume that the state and action spaces are finite and small enough to be tractable. This assumption in turn guarantees the existence of optimal value functions and thus optimal policies[1]. Methods for solving MDPs, which aim at retrieving "the optimal solution" are collectively termed as *exact solution methods*. However, if the assumption of finite problem space is relaxed (*i.e.*, when MDPs are allowed to have infinite state space or action space) exact methods for solving MDPs (generally) cease to exist. This is largely due to the fact that computing exact solutions becomes intractable in the later case.

MDPs with large state-action spaces are solved using *approximate solution methods*. Since exact methods are no-longer tractable, focus is shifted towards finding a near optimal solution by sampling the problem space. These methods aim at retrieving some generalization of the optimal value function based on the limited amount of observed (and rewarded) transitions. The resulting approximation can then be generalized to predict values for states which were not visited during training. Dialogue managers modelled as MDPs often tend to have continuous state spaces (*i.e.*, infinite states). Hence approximate methods for policy optimization play a key role in the dialogue domain.

[1] For example an optimal policy in tabular format lists all possible states along with the best possible action that can be performed from those corresponding states.

This chapter is organized as follows: value function approximation is outlined in 5.1, following which Approximate Dynamic Programming based Fitted Value Iteration is discussed in 5.2. Automatic feature selection scheme, Sparse ADP based sparse fitted-$Q$ and sparse LSPI are discussed in 5.4, 5.5 and 5.6 respectively. Eventually, experimental results on dialogue optimization using ADP and sparse ADP algorithms are presented in 5.7.

## 5.1 Value function approximation

First task in order to employ approximate solution methods is to define some representation for the (approximate) value function. Function approximation schemes can generally be classified as: (i) parametric and non-parametric and (ii) linear and non-linear representations. In this manuscript we focus on linear-parametric representation for value function approximation. A brief summary on parametric function approximation can be found in [Geist & Pietquin, 2010a]. Some examples for linear-parametric representation include: Radial Basis Function (RBF) [Park & Sandberg, 1991], polynomial function [Gergonne, 1974]. Let us now assume that a linear parametric representation of the $Q$-function using a set of features $\phi_{1\dots k}$ is possible. Let $\theta \in \mathbb{R}^p$ represent the parameter or weight vector associated with the features $\phi_{1\dots k}$. The resulting approximate $Q$-function belongs to the hypothesis space $\mathcal{H}$: $\theta \times \phi$. Note that the notation used for the exact value function is $Q$ and the notation for the approximate value function is $\hat{Q}_\theta$:

$$\hat{Q}_\theta = \sum_{i=1}^{k} \theta_i \; \phi_i(s,a) = \theta^T \phi(s,a). \tag{5.1}$$

Once some linear representation of the value function is defined, approximate solution methods can be used to estimate (through the parameter vector $\theta$) a good approximation $\hat{Q}^*_\theta$ of $Q^*$. This manuscript discusses two different classes of approximate solutions: (i) Approximate Dynamic Programming [Bellman & Dreyfus, 1959] (discussed in this chapter) and (ii) Reinforcement learning with function approximation (general case TD RL algorithms as well as those using Kalman Temporal Differences [Geist & Pietquin, 2010b] are discussed in Chapter 6).

Approximate Dynamic Programming(ADP)-based algorithms were introduced for dialogue optimization in [Li *et al.*, 2009; Pietquin *et al.*, 2011b]. These algorithms aim at estimating an approximation of the optimal value function using a fixed set of transition samples $(s, a, r, s')$. The samples required for training are extracted from the limited amount of dialogue corpora. Since ADP algorithms are known to be sample-efficient, dialogue policies can now be optimized directly from the dialogue corpus[1]. This possibility in turn questions the basic necessity of user simulation for the purpose of dialogue optimization. Once an approximate (near optimal) value function is estimated from the corpus the learning terminates and thus the associated policy cannot be improved further. Yet, ADP can be used to retrieve the initial dialogue policy from the corpus. If required, this policy can be improved further using conventional RL algorithms and based on interaction with real users (as summarized in Figure 5.1). Learning by interacting with real users will also result in adapting the policy to any possible changes in user behavior.

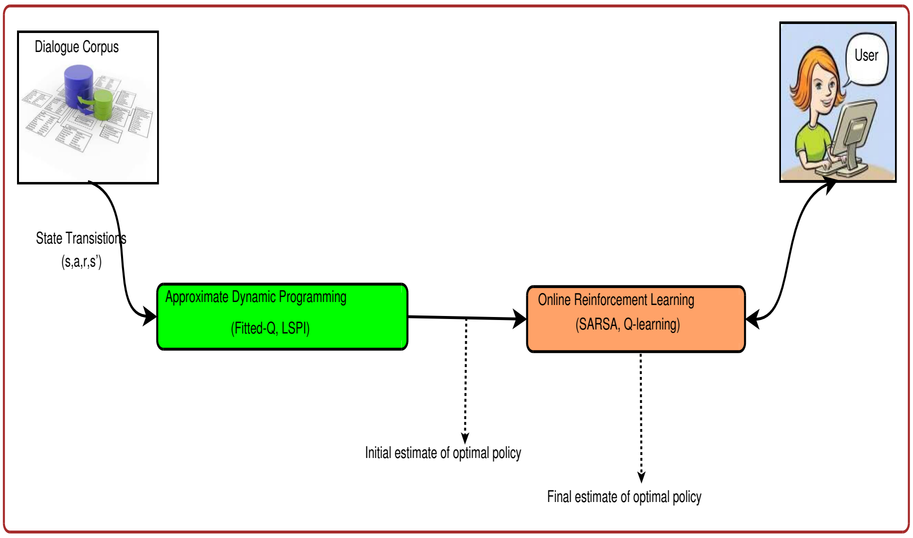

Dialogue Corpus
User
State Transistions
(s,a,r,s')
Approximate Dynamic Programming
(Fitted-Q, LSPI)
Online Reinforcement Learning
(SARSA, Q-learning)
Initial estimate of optimal policy
Final estimate of optimal policy

Figure 5.1: *Policy optimization using both batch and online learning*

The key challenge with regard to approximate solution methods is to choose a

[1] Note that ADP algorithms do not converge if the available corpus is too small in size. See Section 5.7 for more information.

good set of features ($\phi_{1...k}$) such that the the approximated $Q$-function ($\hat{Q}^*$) is not too far from the (true) optimal $Q$-function ($Q*$). Often these features are selected manually with the help of designer's knowledge on the problem (space). One other possibility for feature selection is to employ schemes which can learn this representation automatically from data (for instance, Engel's dictionary [Engel *et al.*, 2004]).

## 5.2 Fitted value iteration

Let us recall the value iteration algorithm discussed in 3.3.2. The optimal action value function $Q^*$ being the fixed point of the Bellman optimality operator $T^*$ can be obtained by iterative contraction, *i.e*, application of the contraction operator to an arbitrary initial value. Being an exact method, value iteration is expected to converge (asymptotically) to the optimal value function:

$$Q^* = T^* Q^*. \tag{5.2}$$

Fitted value iteration (FVI) class of algorithms [Bellman & Dreyfus, 1959; Samuel, 1959] adopts an approximate value iteration scheme[1]. Policy optimization using FVI is very similar to that of the value iteration algorithm. However, the primary difference is the projection operation. The image of $\hat{Q}_\theta$ obtained through the Bellman operator is not guaranteed to be in the space spanned by the basis function, *i.e.*, in $\mathcal{H}$. Thus, it is projected onto the hypothesis space after every iteration. The projection operator is noted as $\Pi$ and defined for any function $f : S \times A \rightarrow \mathbb{R}$ as $\Pi f = \operatorname{argmin}_{\hat{Q}_\theta \in \mathcal{H}} \|f - \hat{Q}_\theta\|^2$. Upon iterating, FVI results in finding a near optimal value function through the parameter vector $\theta$ satisfying:

$$\hat{Q}^*_\theta = \Pi T^* \hat{Q}^*_\theta. \tag{5.3}$$

Fitted-$Q$ (see Figure 5.2) is a specific form of FVI algorithm. It assumes that the $\Pi T^*$ operator is still a contraction and therefore admits the unique fixed point.

[1] Approximate value iteration scheme aims at determining the approximation of an optimal value function

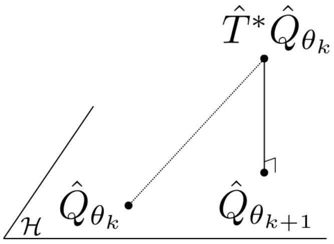


Figure 5.2: Fitted-Q iteration.

The fixed point and therefore the optimal value function ($\hat{Q}^*_\theta$) is searched using an iterative scheme: $\hat{Q}_{\theta_i} = \Pi T^* \hat{Q}_{\theta_{i-1}}$. However, in case of the dialogue domain often the transitions probabilities $T$ and reward function $R$ are not known. In order to cope with this, a *sampled* Bellman optimality operator $\hat{T}^*$ is considered. For instance, given a sample transition $(s, a, r, s')$, it is defined as:

$$\hat{T}^* Q(s,a) = r(s,a) + \gamma \max_{a \in A} Q(s', a). \tag{5.4}$$

Learning from $(s, a, r, s')$ transition samples provides an opportunity to learn a policy from data as shown here. This comes in handy to learn a dialogue policy (directly) from the limited amount of dialogue corpora. Using the definition of $\hat{T}^*$ in Eq 5.3, the fitted-$Q$ algorithm can be defined as:

$$\hat{Q}_{\theta_i} = \Pi \hat{T}^* \hat{Q}_{\theta_{i-1}}. \tag{5.5}$$

At every iteration of Fitted-$Q$, a new estimate of $\hat{Q}_{\theta_i}$ is computed. This estimation depends on how $\hat{T}^* \hat{Q}_{\theta_{i-1}}$ is projected onto the hypothesis space. This process can be seen as a classical regression (which can be performed by any supervised learning algorithm):

$$\hat{Q}_{\theta_i}(s,a) = \Pi \hat{T}^* \hat{Q}_{\theta_{i-1}}(s,a) = \Pi(r + \gamma \max_{a \in A} \hat{Q}_{\theta_{i-1}}(s', a)). \tag{5.6}$$

As shown in Eq. 5.1, let us assume that a linear parametrization of $\hat{Q}_\theta(s,a) = \sum_i \theta_i \phi_i(s,a) = \theta^T \phi(s,a)$ is made available. Let us also assume that a set of transition samples ($\{(s_j, a_j, r_j, s'_j)_{1 \leq j \leq N}\}$) for training purpose is available. One of the several possible means to perform the operation summarized in Eq 5.6 is to perceive it as a least-squares optimization problem as shown below (note that $\phi(s_j, a_j)$ is indeed $\phi_j$):

$$\theta_i = \operatorname*{argmin}_{\theta \in \mathbb{R}^p} \sum_{j=1}^{N} (\hat{T}^* \hat{Q}_{\theta_{i-1}}(s_j, a_j) - \hat{Q}_\theta(s_j, a_j))^2 \tag{5.7}$$

$$= \operatorname*{argmin}_{\theta \in \mathbb{R}^p} \sum_{j=1}^{N} (r_j + \gamma \max_{a \in A}(\theta_{i-1}^T \phi(s'_j, a)) - \theta^T \phi_j)^2 \tag{5.8}$$

$$\theta_i = (\sum_{j=1}^{N} \phi_j \phi_j^T)^{-1} \sum_{j=1}^{N} \phi_j (r_j + \gamma \max_{a \in A}(\theta_{i-1}^T \phi(s'_j, a))) \tag{5.9}$$

The resulting algorithm which employs least-squares optimization as part of fitted-Q is termed as LSFQ. To begin with, an initial parameter vector $\theta_0$ should be chosen (example: with arbitrary values). Similar to value iteration, some stopping criterion ought to be introduced for terminating fitted-$Q$. This criterion can either be some maximum number of iterations or a small difference between two consecutive parameter vector estimations. Let us assume that there are $M$ iterations, the optimal policy is estimated from $\hat{Q}_{\theta_M}$:

$$\hat{\pi}^*(s) = \operatorname*{argmax}_{a \in A} \hat{Q}_{\theta_M}(s, a). \tag{5.10}$$

It may be useful to note that the inverted matrix $(\sum_{j=1}^{N} \phi_j \phi_j^T)^{-1}$ in Eq 5.9 is the same at every iteration. Only the target $r_j + \gamma \max_{a \in A}(\theta_{i-1}^T \phi(s'_j, a))$ depends on the preceding estimate. Thus, the complexity of least squares fitted-$Q$ per iteration is in $O(p^2)$ (ignoring the matrix inversion which remains unchanged).

## 5.3 Least squares policy iteration

Fitted-$Q$ algorithm presented above is an approximate value iteration algorithm. This section presents another ADP algorithm based on approximate policy iteration. Least Squares Policy Iteration (LSPI) [Lagoudakis & Parr, 2003] (similar to policy iteration algorithm discussed in Section 3.3.1) iteratively alternates between policy evaluation and policy improvement. At iteration $(k+1)$, using the approximate value function $(\hat{Q}_{\theta_k})$ estimated in iteration $k$, the policy $\pi_{(k+1)}$ is improved:

$$\pi_{k+1} = \pi_{\text{greedy}}(\hat{Q}_{\theta_k}) \tag{5.11}$$

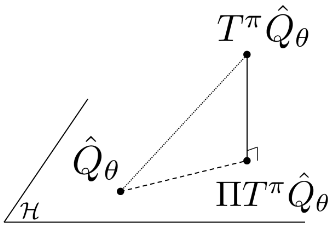


Figure 5.3: LSTD principle.

This policy can be evaluated thanks to the Least Squares Temporal Differences (LSTD) algorithm proposed in [Bradtke & Barto, 1996]. It aims at minimizing the distance between the estimated $Q$-function and the projection of its image through the Bellman evaluation operator onto the hypothesis space (as illustrated on figure 5.3):

$$\theta_\pi = \underset{\theta \in \mathbb{R}^p}{\operatorname{argmin}} \|\hat{Q}_\theta - \Pi T^\pi \hat{Q}_\theta\|^2 \tag{5.12}$$

where $\Pi$ is the projection operator (similar to the one discussed for fitted-$Q$). Generally speaking, $T^\pi \hat{Q}_\theta \notin \mathcal{H}$, thus it is projected back into the hypothesis space, *i.e.*, $\Pi T^\pi \hat{Q}_\theta \in \mathcal{H}$. Here, $N$ available transitions $(s_i, a_i, r_i, s_{i+1}, \pi(s_{i+1}))$ and thus the sampled Bellman operator is considered for policy evaluation. Among the several possible cost functions, one considered in this manuscript takes the form:

$$\theta_\pi = \underset{\theta \in \mathbb{R}^p}{\operatorname{argmin}} \sum_{i=1}^{N} (r_i + \gamma \hat{Q}_\pi(s_{i+1}, \pi(s_{i+1})) - \hat{Q}_\theta(s_i, a_i))^2 \tag{5.13}$$

Equations 5.12 and 5.13 correspond to two nested optimization problems: (i) projecting $T^\pi \hat{Q}_\theta$ to the hypothesis space, and (ii) minimizing the distance between $\hat{Q}_\theta$ and $\Pi T^\pi \hat{Q}_\theta$. Since the parametrization is assumed to be linear, this optimization problem can be solved analytically:

$$\theta_\pi = \left( \sum_{i=1}^{N} \phi(s_i, a_i)(\phi(s_i, a_i) - \gamma\phi(s_{i+1}, \pi(s_{i+1}))^T \right)^{-1} \sum_{i=1}^{N} \phi(s_i, a_i) r_i \qquad (5.14)$$

LSPI was adapted for dialogue optimization in [Li *et al.*, 2009]. However, the feature selection scheme used in this case was based on heuristics which needs careful tuning. The following section details how automatic feature selection schemes can be combined with fitted-$Q$ and LSPI. Both these algorithms are known for their sample efficiency and off-policy aspect. These factors contribute towards direct policy optimization from the dialogue corpus. In particular off-policy aspect provides an opportunity to learn a fairly good policy even if the policy used for generating the samples was not optimal. Thus both fitted-$Q$ and LSPI, make a compelling case for their use in dialogue policy optimization. With regard to computation complexity, fitted-$Q$ and LSPI are $O(p^2)$ and $O(p^3)$ respectively per iteration.

## 5.4 Automatic feature selection

Manually selecting a set of features for value function approximation is a complex task. For instance, using a Radial Basis Function (RBF) [Park & Sandberg, 1991] network for function approximation involves three steps: (i) choose the number of features, (ii) choose the variance $\sigma^2$ of the Gaussian kernels (see Eq. 5.15) and (iii) decide on where each of these kernel centres should be placed. It is often difficult to determine these factors in order to identify a comprehensive[1] set of basis functions. In order to cope with this challenge, an approach which allows learning this representation from data [Engel *et al.*, 2004] is combined with FVI. Combining such a feature selection scheme with LSPI was proposed in [Xu *et al.*, 2007]. This combination will help us to perform sample-efficient

[1] $\hat{Q}_\theta$ will be in the space defined by $\phi$. Thus feature selection directly impacts the quality of approximation.

dialogue optimization without the necessity of manual feature selection. This section provides an overview on how function representation can be learned from data. Data used for this purpose is in the form of state-action pairs $(s, a)$. Let us term $(s, a)$ as $z$ for convenience. Recall that the $Q$-function can be approximated in $\phi$ and $\theta$: $\hat{Q}_\theta(z) = \theta^T \phi(z)$, with $\phi(z)$ being a set of Gaussian kernel[1] basis functions:

$$\phi(z) = \Big(K(z, \tilde{z}_1) \quad \dots \quad K(z, \tilde{z}_p)\Big)^T \tag{5.15}$$

Given a set of training samples or basis $(\{z_1, \dots, z_N\})$ and a kernel K (which comes back to choosing $\sigma$ in case of Gaussian kernel), we will use Engel's dictionary method that aims at: (i) choosing the number of basis functions $(p)$ and (ii) the associated kernel centres $\{\tilde{z}_1, \dots, \tilde{z}_p\} \subset \{z_1, \dots, z_N\}$. An important result with regard to kernels is the Mercer theorem: for each kernel $K$ there exists a mapping $\varphi : z \in Z \rightarrow \varphi(z) \in \mathcal{F}$ ($\mathcal{F}$ being called the feature space) such that $\forall z_1, z_2 \in Z$, $K(z_1, z_2) = \langle \varphi(z_1), \varphi(z_2) \rangle$ (in short, $K$ defines a dot product in $\mathcal{F}$). The space $\mathcal{F}$ can be huge (it is an infinite hyper-sphere for Gaussian kernels), therefore $\varphi$ cannot always be explicitly built. Given this result and from the bi-linearity of the dot product, $Q_\theta$ can be rewritten as follows:

$$Q_\theta(z) = \sum_{i=1}^{p} \theta_i K(z, \tilde{z}_i) = \langle \varphi(z), \sum_{i=1}^{p} \theta_i \varphi(\tilde{z}_i) \rangle. \tag{5.16}$$

Therefore, a kernel-based parametrization corresponds to a linear approximation in the feature space, the weight vector being $\sum_{i=1}^{p} \theta_i \varphi(\tilde{z}_i)$. This is called the *kernel trick*. Consequently, kernel centres $(\tilde{z}_1, \dots, \tilde{z}_p)$ should be chosen such that $(\varphi(\tilde{z}_1), \dots, \varphi(\tilde{z}_p))$ are linearly independent in order to avoid redundancy. In simple words, given a state-action pair from the training basis, Engel's dictionary method [Engel *et al.*, 2004] tries to ensure whether it can be expressed as a linear combination (in the feature space) of elements already present in the dictionary. If it can be expressed so, then the dictionary remains unchanged. However, if it cannot be expressed as a linear combination, then the state-action pair is added

[1] In simple terms kernel functions are mappings or projections to higher dimensional spaces. A two dimensional Gaussian kernel is defined as: $\frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2+y^2}{2\sigma^2}}$, where $\sigma$ is the standard deviation.

to the dictionary. This process is repeated iteratively for all state-action pairs that can be observed in the training corpus. At the end of this iterative process, a set of state-action pairs (those that are present in the computed dictionary) are chosen as basis functions to define the representation of $\hat{Q}_\theta$.

The training base is sequentially processed, and the dictionary is initiated with the first sample: $\mathcal{D}_1 = \{z_1\}$. At iteration $k$, a dictionary $\mathcal{D}_{k-1}$ computed from $\{z_1, \dots, z_{k-1}\}$ is available and the $k^{\text{th}}$ sample $z_k$ is considered. If $\varphi(z_k)$ is linearly independent of $\varphi(\mathcal{D}_{k-1})$, then it is added to the dictionary: $\mathcal{D}_k = \mathcal{D}_{k-1} \cup \{z_k\}$. Otherwise, the dictionary remains unchanged: $\mathcal{D}_k = \mathcal{D}_{k-1}$. Linear dependency can be checked by solving the following optimization problem ($p_{k-1}$ being the size of $\mathcal{D}_{k-1}$):

$$\delta = \underset{w \in \mathbb{R}^{p_{k-1}}}{\operatorname{argmin}} \|\varphi(z_k) - \sum_{i=1}^{p_{k-1}} w_i \varphi(\tilde{z}_i)\|^2 \tag{5.17}$$

Thanks to the kernel trick, this optimization problem can be solved analytically, without computing explicitly $\varphi$. Formally, linear dependency is satisfied if $\delta = 0$. However, an approximate linear dependency is allowed, and $\varphi(z_k)$ will be considered as linearly dependent of $\varphi(\mathcal{D}_{k-1})$ if $\delta < \nu$, where $\nu$ is the so-called sparsification factor. This allows controlling the trade-of between quality of the representation and its sparsity. See [Engel *et al.*, 2004] for details as well as an efficient implementation of this dictionary approach. Let us term ADP performed with automatically selected function representation as Sparse-ADP. FVI and LSPI can be thus be performed with a sparse representation yielding Sparse-FVI and Sparse-LSPI algorithms.

## 5.5 Sparse-Fitted value iteration

Engel's dictionary method is first combined with fitted-$Q$. The aim here is to learn a representation for the $Q$-function using the automatic feature selection scheme in order to perform fitted-$Q$ [1]. In case of the dialogue domain, the trajectories are annotated and stored in the form of transitions $(s, a, r, s')$. In order to compute

[1] Recall the fact that fitted-$Q$ assumes that some representation of $Q$-function is available.

the dictionary, the inputs needed are only state-action pairs $(s, a)$, but not the feedback $(r)$ of the transitions. The transiting state $s'$ is taken into account while considering the next state-action pair and thus ignored while using $(s, a)$.

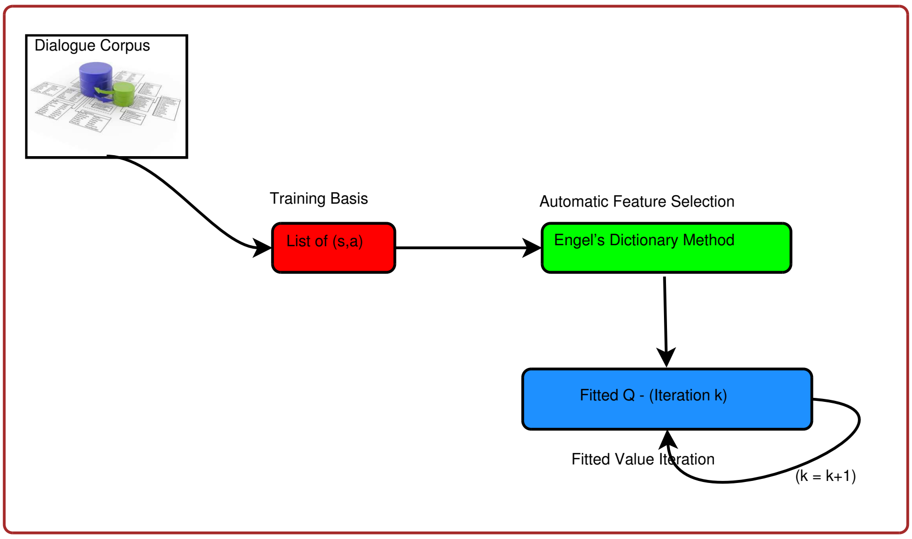


Figure 5.4: *Sparse - FVI (Fitted-Q with feature selection)*

It is important to note that, in case of fitted-$Q$, the input space remains the same over several iterations. Thus the dictionary computed as a pre-processing step from $\{(s_j, a_j)_{1 \le j \le N}\}$ before starting fitted-$Q$ remains unchanged (as shown in Figure 5.4). It can also be noticed that the matrix $(\sum_{j=1}^{N} \phi_j \phi_j^T)^{-1}$ (recall Eq. 5.9) remains unchanged over iterations. Thus it can also be computed during the pre-processing stage of fitted-$Q$. The proposed sparse Least Squares Fitted-$Q$ (sparse LSFQ) algorithm is summarized in Alg. 4.

## 5.6 Sparse-least squares policy iteration

An automatic feature selection scheme was first combined with LSPI in [Xu *et al.*, 2007]. In this section, Engel's method is combined with the LSPI algorithm. The primary difference with regard to sparse LSPI and sparse LSFQ, is the fact that the input space for LSPI changes after every iteration. At iteration $k$, the input is composed of both $(s, a)$ and $(s', \pi_{k-1}(s'))$ state-action couples (as shown in Fig-

**Algorithm 4:** Sparse LSFQ.

```
Initialization
1. Initialize vector $\theta_0$ arbitrarily
2. Choose a kernel $K$ and a sparsification factor $\nu$;
Dictionary computation
Compute the dictionary $\mathcal{D} = \{(\tilde{s}_j, \tilde{a}_j)_{1 \le j \le p}\}$ from $\{(s_j, a_j)_{1 \le j \le N}\}$;
Define the parametrization
$Q_\theta(s,a) = \theta^T \phi(s,a)$ with
$\phi(s,a) = (K((s,a),(\tilde{s}_1,\tilde{a}_1)),\dots,K((s,a),(\tilde{s}_p,\tilde{a}_p)))^T$;
Compute $P^{-1}$
$P^{-1} = (\sum_{j=1}^{N} \phi_j \phi_j^T)^{-1}$;
for $k = 1, 2, \dots, M$ do
    Compute $\theta_k$, see Eq. (5.9);
end
Resulting optimal policy
$\hat{\pi}^*_M(s) = \mathrm{argmax}_{a \in A}\, \hat{Q}_{\theta_M}(s,a)$;
```

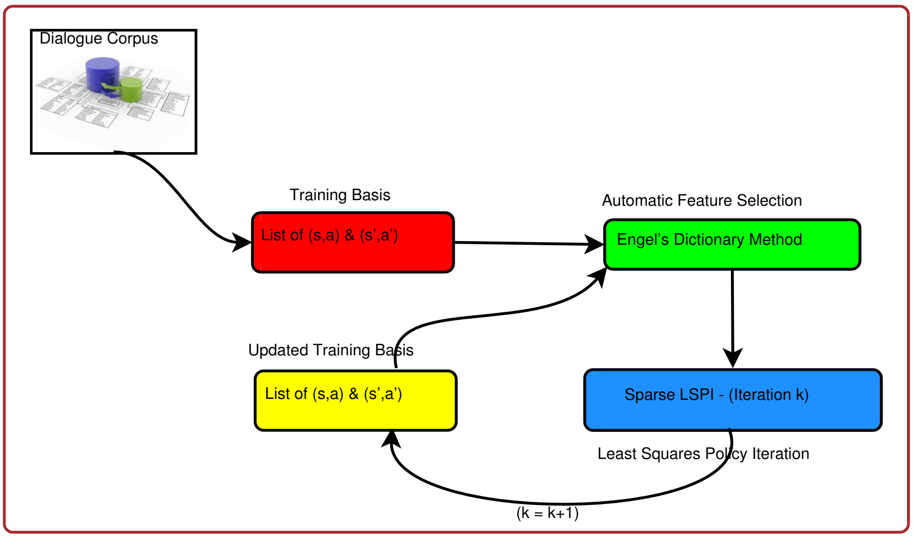

Dialogue Corpus
Training Basis
List of (s,a) & (s',a')
Automatic Feature Selection
Engel's Dictionary Method
Updated Training Basis
List of (s,a) & (s',a')
Sparse LSPI - (Iteration k)
Least Squares Policy Iteration
(k = k+1)

Figure 5.5: *Sparse - LSPI (LSPI with feature selection)*

ure 5.5). This results in a necessity for computing a new dictionary before each iteration using $\{(s,a)_{1\le j\le N},(s',\pi_{k-1}(s'))_{1\le j\le N}\}$. The resulting elements in the dictionary define the parametrization which is considered for $Q$-function evaluation[1]. Sparse LSPI algorithm is summarized in Alg. 5.

**Algorithm 5:** Sparse LSPI.

```
Initialization
1. Initialize policy $\pi_0$
2. Choose a kernel $K$ and a sparsification factor $\nu$;
for $k = 1, 2, \ldots$ do
    Dictionary computation
    Compute the dictionary $\mathcal{D} = \{(\tilde{s}_j, \tilde{a}_j)_{1\le j\le p_k}\}$ from
    $\{(s_j, a_j)_{1\le j\le N}, (s'_j, \pi_{k-1}(s'_j))_{1\le j\le N}\}$;
    Define the parametrization
    $Q_\theta(s,a) = \theta^T \phi(s,a)$ with
    $\phi(s,a) = (K((s,a),(\tilde{s}_1,\tilde{a}_1)),\ldots,K((s,a),(\tilde{s}_{p_k},\tilde{a}_{p_k})))^T$;
    Compute $\theta_{k-1}$, see Eq. (5.12);
    Compute policy $\pi_k$
    $\pi_k(s) = \operatorname{argmax}_{a\in A} \hat{Q}_{\theta_{k-1}}(s,a)$;
end
```

Notice that sparse LSFQ has a lower computational complexity than the sparse LSPI. For sparse LSFQ, dictionary and matrix $P^{-1}$ are computed in a pre-processing step, therefore the complexity per iteration is in $O(p^2)$, with $p$ being the number of basis functions computed using the dictionary method. For LSPI, the inverse matrix depends on the iteration, as well as the dictionary, therefore the computational complexity is in $O(p_k^3)$ per iteration, where $p_k$ is the size of the dictionary computed at the $k^{\text{th}}$ iteration.

## 5.7 Experimental results and analysis

So far in this chapter ADP-based fitted-$Q$, LSPI, sparse LSFQ and sparse LSPI were discussed in detail. The rest of this chapter outlines a set of experiments conducted in the dialogue domain. The primary focus here is to validate (by

[1] Recall policy iteration described in 3.3.1. It is a two step iterative process involving policy evaluation and policy improvement.

means of experimental results) the effectiveness of these algorithms when applied to dialogue policy optimization. Experimental results are analysed in terms of sample efficiency and the quality of retrieved policies. To begin with, the restaurant information SDS with continuous state space, dialogue corpora generation process and choice of $Q$-function representation are elicitated in Section 5.7.1, 5.7.2 and 5.7.3 respectively. Following which dialogue policy optimization for the restaurant information SDS using ADP (see Section 5.7.4) and sparse ADP algorithms (see Section 5.7.5) are discussed in detail.

Recall that ADP algorithms presented in this chapter are batch methods. Here policy optimization is carried out using transitions $(s, a, r, s')$ observed in the dialogue corpus. Using these algorithms it is now possible to directly optimize a policy from fairly limited amount of dialogue data. Unlike dialogue optimization using conventional RL algorithms, these methods does not require a user simulation (since they are sample-efficient) [Pietquin *et al.*, 2011b]. However, to ascertain the quality of the optimized policies, they must be evaluated. Thus user simulation continues to play a critical role at least for the sake of policy evaluation [Georgila *et al.*, 2006]. Even though part of the experiments are conducted using automatically selected features (for Sparse LSFQ and Sparse LSPI), the function approximation method is chosen (manually) to be RBF network. There exist intelligent schemes for model selection in case of batch methods as discussed in [Farahmand & Szepesvári, 2011].

### 5.7.1 Restaurant information system (MDP-SDS)

Restaurant information SDS [Lemon *et al.*, 2006] introduced in chapter 2 is considered again for experimental purposes. However, the state and action space of the MDP-SDS is updated to reflect real world scenarios. The (dialogue) state representation of the MDP-SDS includes confidence scores corresponding to three slots: (1) the location of the restaurant, (2) cuisine of the restaurant and (3) price-range of the restaurant. Each of these confidence scores is the average of slot filling confidence and slot confirmation confidence[1]. Using the such confi-

[1] It may be useful to recall that state of the art statistical ASR and NLU engines also provide a measure of confidence on resulting hypotheses.

dence measures for dialogue management to some extent helps in coping up with the (associated) uncertainty. For instance, if the confidence of filling a slot is very low the dialogue manager can choose to confirm it with the user. However, this is not possible if a binary feature is used for determining whether the slot is filled or not (as suggested in 2.3).

The confidence measure in itself is the probability score ranging from 0 to 1. It is essentially the probability of correct recognition from ASR's perspective or correct interpretation from NLU's perspective. Including such scores in the state representation of the MDP-SDS results in a dialogue problem with three dimensional continuous state space. Thus exact solution methods (such as value iteration for retrieving a tabular policy) cannot be used for policy optimization. With regard to the action space, there exists 13 actions (dialogue-acts): Ask-A-Slot (3 actions, one for each slot), Explicit-Confirm-Slot (3 actions, one for each slot), Implicit-Confirm-And-Ask-A-Slot (6 actions, in combination of 2 slots) and Close-Dialogue action. In order to enable dialogue optimization, some reward function should be defined (as discussed in 3.5). For the experiments presented in this section the following reward function is used:

- reward for correct slot filling: 25;
- penalty for incorrect slot filling: -75;
- penalty for empty slot: -300.

The rewards are given upon reaching the end of the dialogue episode. Note that the reward function includes no time-penalty to penalize the length of dialogue episode explicitly. However, the discount factor $\gamma$ is set to 0.95 to induce an implicit penalty for the length of the dialogue episode. Now that the restaurant information dialogue problem is defined and casted as an MDP, the next step is to solve the MDP-SDS to determine the optimal dialogue management strategy. In order to perform policy optimization using Fitted-$Q$ and LSPI, dialogue data is required. For this reason simulated dialogues are generated using a sub optimal $\epsilon$-greedy policy as explained in the following section. The reason for using simulated data is to showcase the effectiveness of the proposed algorithm on a simple problem. Yet the state space is deliberately chosen to be continuous (infinite states) to elicit the ability of these algorithms to scale up.

### 5.7.2 Dialogue corpora generation

It is important to note that the methods discussed here are off-policy algorithms. Irrespective of the policy used for data generation they can learn an (underlying) optimal policy from the dialogue corpora. This is a key advantage when such algorithms are adapted for policy optimization in the dialogue domain. In order to showcase this ability, dialogues are generated using an inferior policy. Also it is important to note that the samples used for learning should be explorative enough to learn a good generalization or approximation of the value function. Thus, the sub-optimal $\epsilon$-greedy policy used for dialogue corpus generation is a combination of a hand-crafted baseline policy and a uniform policy. Here, for each decision, the baseline policy is chosen with a probability $(1 - \epsilon)$ and the random (uniform) policy is chosen with a probability $\epsilon$. To begin with, a baseline system which uses a hand-crafted policy is built. The hand-crafted policy tries to fill the three available slots one after the other and terminates the dialogue episode if all the slots are filled. Perhaps this is one of the simplest way to perform the restaurant-info dialogue task. The sub-optimality of the hand-crafted policy, used here for data generation, comes from the fact that confirmatory system actions are not used.

Dialogues generated using a hand-crafted policy alone will not provide a good sampling of the problem space. Thus, it is combined with a pure random or uniform policy using $\epsilon$-greedy approach. The value of $\epsilon$ is set as 0.9 during dialogue generation process. By using the random policy in conjunction with a hand-crafted policy it is ensured that the problem space is sampled better and at the same time there are also episodes which have a successful task completion reward. In order to generate simulated dialogues, an n-gram user simulation is plugged into the DIPPER dialogue management framework [Lemon *et al.*, 2006]. The $\epsilon$-greedy policy is then used to control the interaction between the DIPPER dialogue manager and simulated user for dialogue simulation. Using this set-up, 56k dialogue episodes resulting in 400k (approximately) dialogue turns (state transitions) are collected.

### 5.7.3 Q-function representation

In order to use ADP for policy optimization, the first step is to define the representation of the action-value function. To begin with, manual feature selection scheme for parametric representation of the $Q$-function is considered (as described in Eq 5.1). For the experiments, an RBF network based representation of the $Q$-function is employed (see Eq 5.15). The RBF network has 3 Gaussians for every dimension in the state space and considering that the restaurant-info problem has 13 actions, a total of 351 (*i.e.,* $3^3 \times 13$) features are used for representing the $Q$-function. Gaussians are centred at $\mu_i = 0.0$, $0.5$, $1.0$[1] and with a standard deviation $\sigma_i = \sigma = 0.25$.

### 5.7.4 Policy optimization using ADP

To begin with, the initial set of experiments (presented in this section) focuses on using the manually defined $Q$-function representation. Fitted-$Q$ [Chandramohan *et al.*, 2010a] and LSPI [Li *et al.*, 2009] are used to perform dialogue policy optimization. The primary focus of these experiments is to explore the sample efficiency of ADP algorithms when applied to dialogue optimization. We know that these algorithms learn from transition samples, *i.e.*, $(s, a, r, s')$. The available training data set is divided into 8 smaller data sets each with 50k dialogue turns. LSPI and fitted-$Q$ based policy optimization is performed 8 different times using the 8 different data sets. During each of these training sessions the number of samples used for training is varied from 5k to 50k. Resulting $Q$-functions after convergence while using 5k, 10k, 20k, 30k, 40k, 50k samples are stored in memory. Using these $Q$-functions, optimal policies (greedy with respect to $Q$-function) are computed.

The stopping criterion for LSPI is based on the number of changes made to the policy being optimized. The learning is terminated if the number of changes to the policy is less than $\xi$ (during the experiments the value of $\xi$ is set to 1).

[1] Recall that dialogue state involves three dimensions with values ranging from 0 to 1.

The stopping criterion for the fitted-$Q$ is as follows: learning is terminated at iteration $n$, if $\|\theta_n - \theta_{(n-1)}\|_1 < \xi$, where $n$ is the iteration number and $\xi$ is the convergence threshold.

The estimated policies are evaluated based on the quality of dialogues generated by the DIPPER dialogue management framework (controlled by resulting policies) and an n-gram user simulation. In total 100 dialogue episodes are generated using each of the 8 policies at every stage (for instance while using 5k samples for training). The average discounted sum of rewards (and their standard deviation) obtained by the policies trained using varied number of samples are shown in Figure 5.6 (in semi-log scale). This figure indicates that both Fitted-$Q$ and LSPI are sample-efficient algorithms for dialogue policy optimization. These approaches can learn good dialogue policies (compared to the baseline hand-crafted policy) from limited number of samples contrary to other RL algorithms which are known to be data intensive [Pietquin *et al.*, 2011b] (such as $Q$-learning)[1]. Even though the performance of fitted-$Q$ and LSPI are more or less the same, LSPI-based policy optimization comes at an additional computational cost considering the fact that it needs matrix inversion at each iteration, contrary to fitted-$Q$ (in which matrix inversion is done only once).

### 5.7.5 Dialogue optimization using Sparse ADP

Even though ADP algorithms for dialogue optimization are sample-efficient, they need the representation of $Q$-function to be defined beforehand. For this reason we combine automatic feature selection schemes with LSPI and fitted-$Q$ algorithms (as discussed in sections 5.5 and 5.6). Similar to the experiments with ADP algorithms, sparse LSFQ and sparse LSPI are used to optimize the dialogue policy of the restaurant information dialogue system. These algorithms learn a representation of the $Q$-function and then retrieve the estimated optimal $Q$-function from the data. The number of features selected by the dictionary scheme varies depending on the value $\nu$ (which is the sparsity coefficient). During

[1] Experimental evidence showcasing the sample inefficiency of SARSA and $Q$-learning is presented in chapter 6.

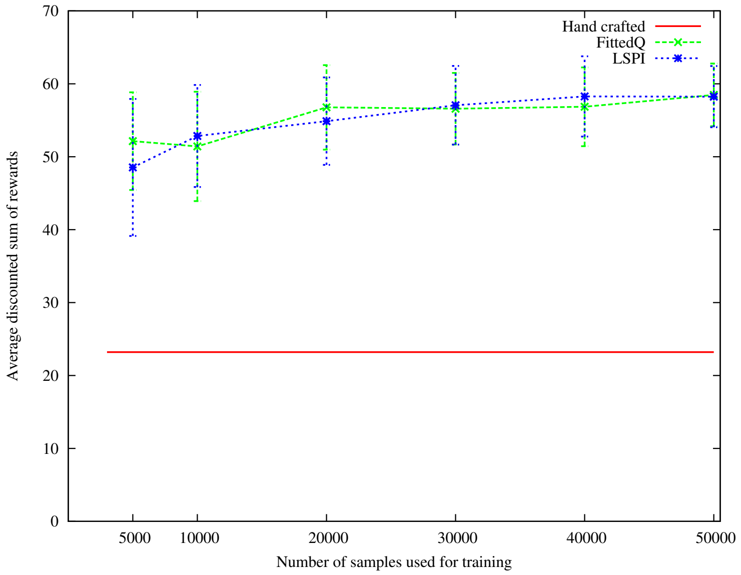


Figure 5.6: *Evaluation of policies learned using Fitted-Q and LSPI*

the experiments $\nu$ is first set to 0.7 and then to 0.8[1]. Dictionaries computed with $\nu$=0.7 and $\sigma_i = \sigma = 0.25$ had about 325 to 365 features depending on the number of samples used for learning. Dictionaries computed with $\nu$=0.8 and $\sigma_i = \sigma = 0.25$ had about 286 to 317 features depending on the number of samples used.

As discussed earlier, the dictionary is computed only once at the beginning of sparse LSFQ-based learning. In this case it is computed using $(s_i, a_i)_{1\leq i\leq N}$ couples from the dialogue corpora. In case of sparse LSPI the dictionary is computed before each iteration using both $(s_i, a_i)_{1\leq i\leq N}$ and $(s'_i, \pi_k(s'_i))_{1\leq i\leq N}$ couples. Similar to the process explained in Section 5.7.4, multiple learning sessions are performed using the eight different data sets. The stopping criterion for sparse LSFQ is same as that of fitted-$Q$ whereas the stopping criterion for sparse LSPI has $\xi$ set to 500. In the later case, since the basis function of the $Q$-function is updated before each iteration, the convergence criteria is relaxed (based on trail

[1] Recall that for higher values of $\nu$, the resulting representation will be sparser as well as cruder. For Gaussian kernels, dictionary computation with $\nu$ set to 1, will result in having only one element in the dictionary.

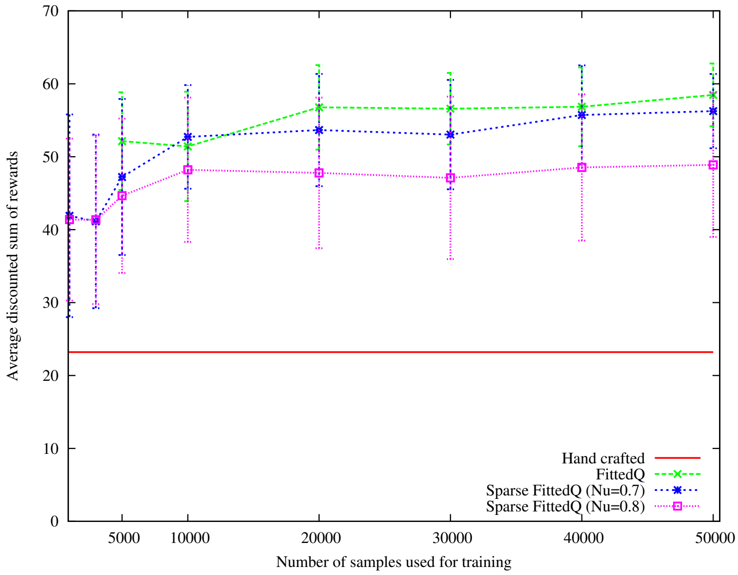


Figure 5.7: *Evaluation of policies learned using Fitted-Q and Sparse Fitted-Q*

and error to get good performance).

In order to evaluate the resulting dialogue policies, synthetic dialogues are generated using DIPPER and an n-gram user simulation. Policies are evaluated based on their average discounted sum of rewards. Comparison between the policies learned using fitted-$Q$ and sparse LSFQ, is presented in Figure 5.7. It can be observed that sparse LSFQ is also sample-efficient and indeed can learn good dialogue policies (some times as good as fitted-$Q$) compared to the baseline policy. It can be observed that the performance of fitted-$Q$ is slightly better when compared to sparse Fitted-$Q$. This may be due to the fact that the hand selected features for this specific dialogue problem seem to be a good set of features. However, it is important to recall that manually selecting a good feature set may not be always a possible option (especially when dealing with larger dialogue problems).

Comparison between LSPI and sparse LSPI policies is presented in Figure 5.8. It can be observed that policies learned using sparse LSPI performed significantly better than the baseline hand-crafted policy. But when compared to LSPI the

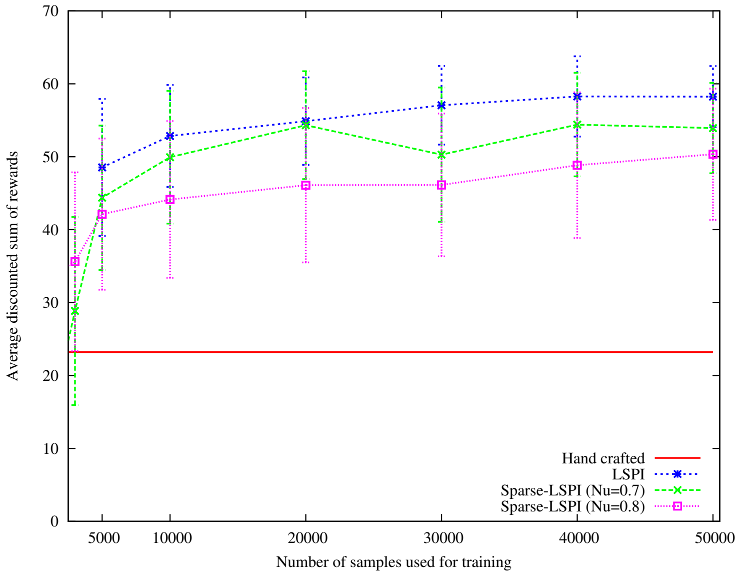


Figure 5.8: *Evaluation of policies learned using LSPI and sparse LSPI*

policies are moderately inferior. This may partially be attributed to the convergence criteria used for stopping the learning. For instance, hand selecting the policy with the least number of policy changes may result in better performance. Similar to fitted-$Q$, in this case as well the good performance of LSPI is largely due to the quality of manually selected features. When the complexity of the problem grows, the possibility for manual feature selection may cease to exist or may become intractable. Even though the results of ADP and Sparse ADP are fairly comparable, the policies estimated using Sparse ADP seem to be less stable (since they have a large variance) when compared to ADP policies.

The following dialogue is an example interaction between the DIPPER dialogue manager (controlled by a policy learned using Fitted-Q with 10k samples) and the n-gram user simulation. Here it can be observed that the learnt dialogue policy not only fills the slots with user preferences (like the baseline policy) but also confirms the slot using explicit and implicit confirmatory actions.

```
=========================================================
Dialogue between DIPPER dialogue manager and n-gram user:
  Dialogue State: [0, 0, 0]
    Dialogua Act: askASlot-slot_3
  Dialogue State: [0, 0, 0.5]
    Dialogua Act: explicitConfirm-slot_3
  Dialogue State: [0.368238, 0, 1]
    Dialogua Act: implConfAskASlot-slot_1_ASK_slot_2
  Dialogue State: [0.917882, 0.499996, 1]
    Dialogua Act: explicitConfirm-slot_2
  Dialogue State: [0.958942, 0.5, 1]
    Dialogua Act: explicitConfirm-slot_2
  Dialogue State: [0.958942, 1, 1]
    Dialogua Act: closingDialogue-null
=========================================================
Intended goal of the simulated user:
  Type: Indian
  Price-range: Cheap
  Location: City-Centre
User goal captured by the dialogue manager:
  Type: Indian
  Price-range: Cheap
  Location: City-Centre
=========================================================
Dialogue statistics:
  ActualReward: 75 (for successful task completion)
  Dialogue length: 6
  Discounted reward at end of the episode: 55.1319
=========================================================
```

*Example dialogue episode from the restaurant-information MDP-SDS*

Experimental results summarized in this chapter [Chandramohan *et al.*, 2010a,b; Li *et al.*, 2009; Pietquin *et al.*, 2011b] outlines an exciting direction of work with regard to policy optimization in the dialogue domain. Firstly, the off-policy aspect of ADP algorithms comes in handy for direct policy optimization from dialogue corpora. Secondly, the sample-efficiency of these algorithms means that user simulators are no longer required for the sake of policy optimization. Even though user simulators still play a role in policy evaluation, alternative schemes for evaluation based on feedback from real users tends to be more reliable.

# Chapter 6

# Kalman Temporal Differences

Online reinforcement learning-based dialogue optimization is discussed in this chapter. Let us recall the taxonomy of RL algorithms discussed in Section 3.4.4. Online RL algorithms make use of samples obtained during live interaction with the real users for dialogue optimization. Based on the differences between the policy being evaluated and policy used for control, these algorithms can be classified into on-policy and off-policy algorithms. In the case of off-policy algorithms, the policy used for data generation is different from the policy associated to the $Q$-function being estimated, thus learning can be performed in a off-line setting using a fixed data set in the form of a dialogue corpus.

Standard RL algorithms such as SARSA (see section 3.4.2) and $Q$-learning (see section 3.4.3) are based on Temporal Difference (TD) estimates[1]. These algorithms in particular are known to have poor sample efficiency and thus need vast amounts of data for policy optimization. Yet, they are the most frequently used algorithms for policy optimization in the dialogue domain. More recently, Kalman Temporal Differences (KTD) [Geist & Pietquin, 2010b] based value function approximation was introduced. KTD in itself is a generic framework which casts value function approximation task as a filtering problem. It can be used to retrieve specific flavours of algorithms such as KTD-$Q$ (off-policy) and KTD-SARSA (on-policy). This chapter is organized as follows: policy optimization using online/off-policy are discussed in sections 6.1 ($Q$-learning) and  6.2 (KTD-

[1] Recall from section 3.4.1, TD is the commonly used learning scheme in RL, where the current estimate of $Q(s, a)$ is updated using the future estimate i.e., $Q(s', a')$.

$Q$), following which policy optimization using online/on-policy algorithms are outlined in section 6.3 (SARSA and KTD-SARSA). Eventually, section 6.5 outlines the effectiveness of these algorithms when applied for dialogue optimization.

## 6.1 Q-learning with function approximation

Q-learning with function approximation is one of the classic RL algorithm used for optimizing large or continuous state MDPs. Assume for now that the optimal state-action values ($Q^*(s_j, a_j) = q_j^*$) can be observed. Once these values are (assumed to be) known, we can treat the value function approximation problem as a regression problem. There exists several possible cost functions in order to minimize the associated (regression) error. The cost function considered here takes the form:

$$\theta_i = \underset{\theta \in \mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} (q_j^* - \hat{Q}_\theta(s_j, a_j))^2. \tag{6.1}$$

The goal here is to determine the parameter vector $\theta_i$ which assures a good approximation of the optimal value function. In the supervised learning domain, it is common to minimize the associated cost function using methods such as gradient descent. Let us assume that the typical gradient descent scheme is applied for cost minimization. In this case the update rule upon observing $(s_j, a_j)$ and $q_j^*$ can be defined as follows:

$$\theta_i = \theta_{i-1} + \alpha_i \nabla_\theta \hat{Q}_{\theta_{i-1}}(s_i, a_i)(q_j^* - \hat{Q}_{\theta_{i-1}}(s_i, a_i)). \tag{6.2}$$

Here, $\alpha_i$ is the step size or the learning rate. Revisiting the assumption made before, it is important to note that the optimal state-action values indeed are not observable and only an estimate of this optimal value is available. Because of this, $q_i^*$ is bootstrapped *i.e.*, it is replaced by an estimate of this optimal $Q$-value. Recall that the optimal $Q$-function is the unique fixed-point of the Bellman optimality operator. Thus, a good estimate for $q_j^*$ is:

$$\hat{T}^* \hat{Q}_{\theta_{i-1}}(s_i, a_i) = r_i + \gamma \max_{a \in A} \hat{Q}_{\theta_{i-1}}(s_{i+1}, a). \tag{6.3}$$

Using the estimation of $q_j^*$ (see Eq. 6.3), we can now rewrite Eq. 6.1 as:

$$\theta_i = \theta_{i-1} + \alpha_i \nabla_\theta \hat{Q}_{\theta_{i-1}}(s_i, a_i)(r_i + \gamma \max_{a \in A} \hat{Q}_{\theta_{i-1}}(s_{i+1}, a) - \hat{Q}_{\theta_{i-1}}(s_i, a_i)). \tag{6.4}$$

As it can be observed in Eq. 6.4, the problem being dealt with is treated as a first order optimization problem. The update rule used here is the typical gradient ascent (*i.e.*, $\nabla_\theta$ is utilised). The convergence criteria associated with gradient ascent-based optimization is asymptotic and thus requires vast amount of data to estimate the optimal $Q$-function. However, dialogue corpora generation is an expensive process. Thus, user simulations are employed to generate synthetic dialogues. It may be useful to recall that, $Q$-learning estimates the optimal $Q$-function using samples generated by another control policy (*i.e.*, off-policy algorithm). Thus, dialogue transitions collected using some sub-optimal policy can in turn be used to optimize a policy using $Q$-learning in a offline and off-policy setting.

## 6.2 Kalman Temporal Differences

Kalman Temporal Differences (KTD) framework [Geist *et al.*, 2009] aims at estimating the approximate value function. Here the task of estimating this function is casted as a filtering problem, and solved using Kalman filters [Kalman, 1960]. The advantage of using Kalman filters for value function approximation comes from its ability to predict statistically optimal estimates (best linear predictor in a least-means squares sense), in-spite of receiving non-stationary and noisy inputs. More importantly, Kalman filters can estimate optimal values in a sample efficient manner (since it uses second order statistics). Thus KTD-based algorithms, when employed for dialogue policy optimization, can effectively handle non stationary user behaviors and optimize policies from limited amount of data. KTD is a generic framework for value function approximation and so specific versions of algorithms such as KTD-$Q$ and KTD-SARSA can be derived. To begin with, this section explains value function approximation using Kalman filters and outlines the KTD-$Q$ algorithm.

### 6.2.1 KTD-$Q$ - online/off-policy algorithm

A filtering problem involves predicting the best estimate of some unknown variables using potentially noisy observations, provided a generative model linking the observations to some unknown variables. Kalman filters[Kalman, 1960] are well known algorithms for solving such problems. Let us consider a typical prediction problem where $X$ has to be estimated using some related observations $Y$. To begin with, $X$ and $Y$are modelled as random variables. Now the solution for this problem can be summarized as finding a conditional expectation of $X$ in terms of $Y$, *i.e.*, $\hat{X}_{i|i} = E[X|Y_1, \dots, Y_i]$. The Kalman-filter-based solution requires determining two factors: (i) how the hidden quantities evolve? and (ii) how they are linked to observations? (see Eq. 6.5). These factors together are often termed as a state-space formulation.

$$\begin{cases} X_{i+1} = f_i(X_i, v_i), \\ Y_i = g_i(X_i, w_i). \end{cases} \tag{6.5}$$

Here the equation used to determine the evolution of $X_{i+1}$ is called the *evolution equation* and the one used to determine $Y_i$ is called as the *observation equation*. Given these equations, the Kalman filter proposes a linear update rule providing the best linear estimate of the conditional expectation $\hat{X}_{i|i} = E[X|Y_1, \dots, Y_i]$:

$$\hat{X}_{i|i} = \hat{X}_{i|i-1} + K_i(Y_i - \hat{Y}_{i|i-1}) \tag{6.6}$$

where $K_i$ is the so-called Kalman gain and $\hat{Y}_{i|i-1}$ is the prediction of the observation given the current estimate of $\hat{X}_{i|i-1}$ and Eq. 6.5.

$Q$-function approximation given its equivalence to the filtering problem can be solved using Kalman filters. However, in case of dialogue systems the parameters can evolve with time (in which case the hidden variable is the $\theta$ vector). This variation is primarily due to constant variation in user behavior. Since the optimal policy is non stationary, how $\theta$ evolves over a period of time cannot be determined, *i.e.*, the function $f$ of the evolution equation is not known. Thus, a random walk evolution model is assumed for parameters ($\theta_i = \theta_{i-1} + v_i$ with $v_i$ being a centered white noise, called the evolution noise). Rewards are linked to parameters through the (sampled) Bellman optimality equation (see Eq. 6.3) which provides the $g_i$ function of Eq. 6.5. A centered and white observation noise $n_i$ is added because the estimated value $Q_\theta$ does not necessary lie in the functional space $\mathcal{H}$ (inductive bias, which is the difference between the $Q$-function and its estimate in $\mathcal{H}$.). This provides the following state-space formulation in the Kalman filtering paradigm:

$$\begin{cases} \theta_i = \theta_{i-1} + v_i \\ r_i = \hat{Q}_{\theta_i}(s_i, a_i) - \gamma \max_{a \in A} \hat{Q}_{\theta_i}(s_{i+1}, a) + n_i \end{cases} \tag{6.7}$$

The KTD-$Q$ algorithm provides the best linear estimator. The Kalman estimator minimizes the expectation of the mean-squared error conditioned on past observed rewards: $J_i(\theta) = E[\|\theta_i - \theta\|^2 | r_{1:i}]$. Here both parameters $\theta_i$ and the associated variance $P_i$ are maintained. Upon observing a new transition, $P_{r_i}$ and $P_{\theta r_i}$ as well as the predicted reward $\hat{r}_i$ are computed. Note that $\hat{r}_i \approx E[\hat{Q}_{\theta_{i-1}}(s_i, a_i) - \max_{a \in A} \hat{Q}_{\theta_{i-1}}(s_{i+1}, a)]$ is indeed the temporal difference error. These values are used to update the Kalman gain and thus $\theta_i$ and $P_i$ (variance associated with $\theta$) as shown here:

$$\begin{cases} K_i = P_{\theta r_i} P_{r_i}^{-1} \\ \theta_i = \theta_{i-1} + K_i(r_i - \hat{r}_i) \\ P_i = P_{i-1} - K_i P_{r_i} K_i^T \end{cases} \tag{6.8}$$

It is important to note that the computation of the Kalman gain relies on second order statistics computable from data. This approach is sample efficient when compared to $Q$-learning which makes use of first order estimates. The sample efficiency is validated using experimental results in Section 6.5 where KTD-based algorithms are compared to $Q$-learning with function approximation. Non-linearity due to non-stationary behavior of users can be handled effectively using Kalman filtering using unscented transform (a sampling technique used to choose a minimal set of samples). This ability of unscented Kalman filtering comes in hand when applied for dialogue optimization in the form of value function prediction or estimation.

## 6.3 SARSA with function approximation

SARSA another classic RL algorithm discussed in Section 3.4.2 aims at estimating $Q$-functions for MDPs with finite problem space. However, in order to deal with large or continuous space MDPs, only approximate solutions are feasible. In case of SARSA, estimated value of $\theta_i$ is updated upon observing a dialogue transition of the form $(s_i, a_i, r_i, s_{i+1}, a_{i+1})$. It may be useful to note that the action $a_{i+1}$ is chosen by the control policy used for user interaction[1]. SARSA with function approximation is similar to the $Q$-learning algorithm explained in Section 6.1. Recall that in case of $Q$-learning, the optimal $Q$-function is estimated using the sampled Bellman optimality operator. Whereas, the estimate used with regard to SARSA is based on the the sampled Bellman evaluation operator $\hat{T}^\pi$. Thanks to the function approximation-based update rule which can be derived as discussed in Section 6.1. The update rule for SARSA with function approximation in order to estimate $\theta_i$ can be obtained in a similar way:

$$\theta_i = \theta_{i-1} + \alpha_i \nabla_\theta \hat{Q}_{\theta_{i-1}}(s_i, a_i)(r_i + \gamma \hat{Q}_{\theta_{i-1}}(s_{i+1}, a_{i+1}) - \hat{Q}_{\theta_{i-1}}(s_i, a_i)) \tag{6.9}$$

### 6.3.1 KTD-SARSA - online/on-policy algorithm

Another specific form of the KTD framework can be obtained to perform online/on-policy dialogue optimization. We know that in case of SARSA, value estimates are indeed obtained using the sampled Bellman evaluation operator. Thus, for KTD-SARSA the update rule is derived from the following state-space formulation:

$$\begin{cases} \theta_i = \theta_{i-1} + v_i \\ r_i = \hat{Q}_{\theta_i}(s_i, a_i) - \gamma \hat{Q}_{\theta_i}(s_{i+1}, a_{i+1}) + n_i \end{cases} \tag{6.10}$$

[1] Recall that in case of online/on-policy learning, both the control policy and the policy being estimated are one and the same.

## 6.4 Uncertainty management in KTD

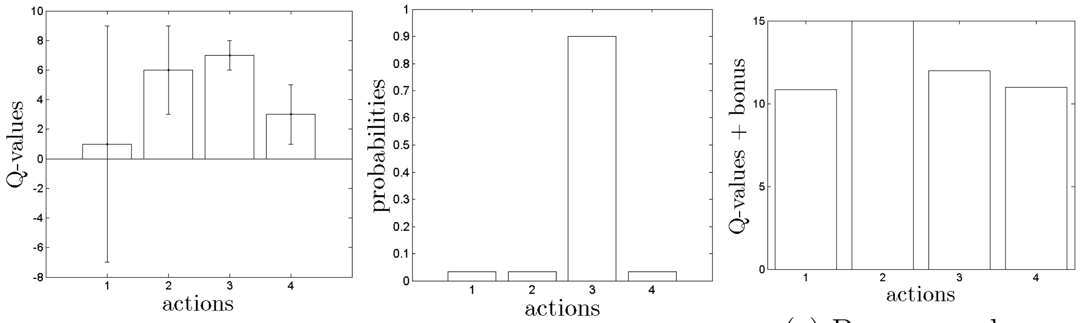
On-policy reinforcement learning algorithms suffer from the exploration vs exploitation dilemma. Dialogue manager interacting with an user must decide whether to choose dialogue acts based on the experience it gained from past interactions (*i.e.*, exploitation) or to explore the outcome of dialogue acts which were not used during past interactions (*i.e.*, exploration). As discussed in Section 3.4, one possible way to cope with this challenge is to employ an $\epsilon$-greedy action selection scheme[1]. In order to efficiently and quickly optimize a dialogue policy, online policy learners must have a good trade-off between exploration and exploitation. Thus, choosing a right $\epsilon$ is of critical importance with regard to policy optimization. There exist several methods for choosing $\epsilon$ in order to cope with this dilemma. Yet most of these methods rely heavily on heuristics in one form or another.

In recent years, smarter ways to cope with the exploration vs exploitation dilemma have been studied with significant interest in the RL domain. For instance, using uncertainty information (associated with $Q(s, a)$ estimates) has been shown to be effective [Dearden *et al.*, 1998; Kolter & Ng, 2009; Sakaguchi & Takano, 2004; Strehl & Littman, 2006]. Most of these approaches are designed for finite space MDPs with tabular case[2]. Even though these methods are effective, extending them for infinite or continuous space MDPs (such as dialogue management MDP) is often impractical.

KTD framework provides an opportunity (both for finite and continuous space MDPs) to cope with this challenge by computing uncertainty information associated with the estimates of state-action values [Geist *et al.*, 2008]. Let us recall the KTD update rule summarized in Eq. 6.8. Here apart from updating $\theta_i$, the variance $P_i$ associated with $\theta_i$ estimate is also computed. Using these quantities, it is possible to compute, for any state-action couple $(s, a)$, the value estimate $\hat{Q}_{\theta_i}(s, a)$, as well as the associated variance $\sigma^2_{Q_i}(s, a)$. In simple terms this vari-

[1] The agent explores the problem space by choosing a random action with probability $\epsilon$ and exploits the experience gained (summarized in the form of $Q$-function) by choosing greedy actions with probability (1 - $\epsilon$)

[2] In case of finite space MDPs, uncertainty information can be computed by using state-action visitation frequencies.



(a) Q-values and variance

(b) $\epsilon$-greedy.

(c) Bonus-greedy.

Figure 6.1: Q-values and policies.

ance measure is the algorithm's confidence on its estimate of $\hat{Q}_{\theta_i}(s,a)$. Using this variance active learning schemes can be formulated as shown in [Geist & Pietquin, 2010c].

The rest of this section explores the possibility of using an intelligent exploration scheme for dialogue policy optimization. For this purpose, the bonus-greedy policy proposed in [Kolter & Ng, 2009] is employed. It consists in acting greedily with respect to the estimated $Q$-function plus some bonus ($\beta$ and $\beta_0$ being two parameters to be set by the system designer):

$$a_{i+1} = \operatorname*{argmax}_{a \in A} \big(\hat{Q}_{\theta_i}(s_{i+1},a) + \beta \frac{\sigma^2_{Q_i}(s_{i+1},a)}{\beta_0 + \sigma^2_{Q_i}(s_{i+1},a)}\big) \tag{6.11}$$

To begin with, during the initial stages of learning, the estimated variance is expected to be high and so the bonus will be approximately equal to $\beta$. However, as learning progresses, the variance is expected to decrease[1]. In the final stages of the learning process, when state-action values are well estimated, the variance tends to 0. Thus, the bonus greedy policy becomes totally greedy with respect to the estimated state-action value function.

Action selection using bonus and $\epsilon$-greedy policies is explained here with an example [Geist, 2009]. Let us consider an arbitrary $Q$-function for some given state and 4 different actions as shown in Figure 6.1 (a). This shows the $Q$-values for each of the 4 actions along with the associated uncertainty in the form

[1] After several observations the algorithm will grow in confidence and hence the variance will decrease.

of standard deviation. It can be noted that action 3 has the highest $Q$-value and also the lowest uncertainty, while action 1 has the lowest $Q$-value but the highest uncertainty. The probability distribution associated to $\epsilon$-greedy policy (action selection scheme) is shown in Figure 6.1 (b). In this case action 3 has the highest probability (for being selected) while other actions have some uniform (low) probability. The probabilities of other actions are the same despite the fact that these actions have different $Q$-values and associated variances. The bonus-greedy action selection is shown in Figure 6.1 (c). In this case the score for action selection is computed as explained in Eq 6.11. Both arbitrary $Q$-values and associated standard deviations are used with regard to bonus-greedy. Thus, action 2 becomes the best action to be chosen. It can also be noticed that the rest of the actions have approximately the same score, despite the fact that they have quite different $Q$-values.

## 6.5 Experimental results and analysis

So far in this chapter, different online reinforcement learning algorithms were discussed. The primary motivation to introduce commonly used RL algorithms (such as $Q$-learning and SARSA) along with relatively new KTD-framework-based algorithms is to provide an unified view on these algorithms. The rest of this chapter focuses on comparing the effectiveness of using these algorithms for dialogue policy optimization. The dialogue problem studied in this section is the same restaurant information dialogue system. For this reason, restaurant information MDP-SDS formulation discussed in Section 5.7 is reused.

### 6.5.1 Online/off-policy dialogue optimization

To begin with, experiments are first conducted using two algorithms: (i) $Q$-learning and (ii) KTD-$Q$. Since they are off-policy algorithms, dialogue corpora generated using sub-optimal policies can be used for training. During the experiments both prior variance and observation noise of KTD-$Q$ are set to 1. Learning rate $\alpha$ is set to 0.2. Both KTD-$Q$ and $Q$-learning employed the *same parametrized*

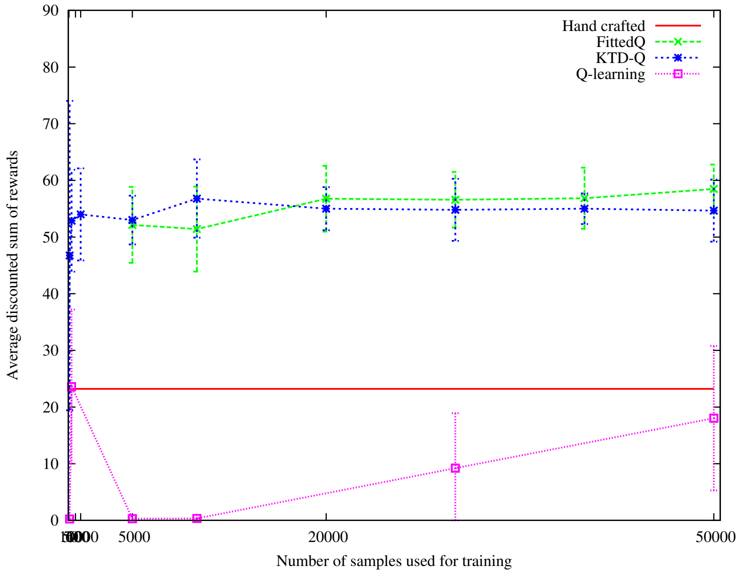


Figure 6.2: *Evaluation of KTD-Q (linear scale)*

*representation of the Q-function*[1]. Using the 400K training samples, policy optimization is performed in eight parallel sessions. Resulting dialogue policies are used in conjunction with the DIPPER dialogue management engine with an n-gram user simulation, in order to generate synthetic dialogues for evaluation purposes. A hand-crafted system is used as the baseline for evaluation.

Performance of the policies learned using KTD-$Q$ and $Q$-learning is presented in Figure 6.2 (in linear scale). It can be observed that KTD-$Q$ policies outperform the baseline (hand-crafted) system as well as $Q$-learning policies (quite significantly). Figure 6.3 (in semi-log scale) shows that KTD-$Q$ can learn good policies from few hundred samples when compared to $Q$-learning. The sample efficiency of KTD-$Q$ showcased here is very good. This encourages policy optimization using KTD-$Q$ especially in the dialogue domain (where corpora generation is known to be expensive and time consuming process).

On the contrary, $Q$-learning, being data intensive, fails to learn a good policy

[1] Same $Q$-function representation is employed in order to compare algorithms under a unified setting.

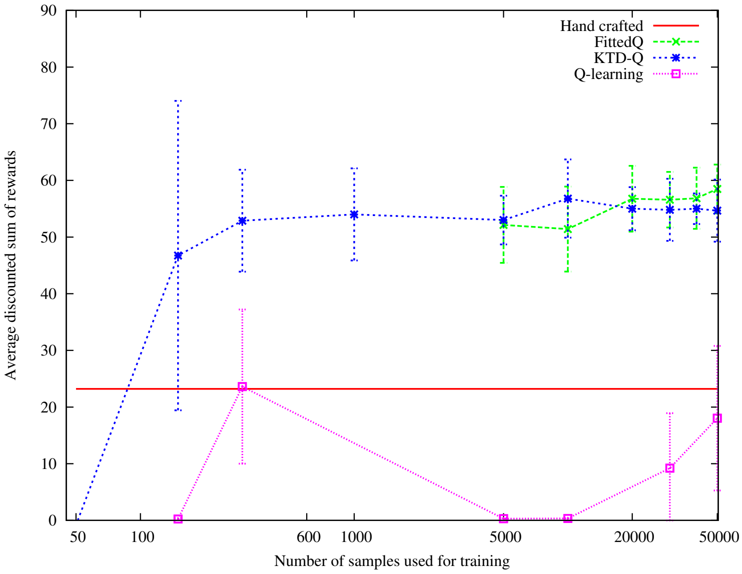


Figure 6.3: *Evaluation of KTD-Q (semi-log scale)*

even after using 50K samples. Poor performance of $Q$-learning may be attributed to its value estimation scheme (which uses first order statistics). Also it can be observed that the policies learned using KTD-$Q$ are nearly as good as the policies learned using Fitted-$Q$. However, choosing KTD-$Q$ parameters can be tricky task at times. Thus it is ideal to perform policy optimization using a cross-validation scheme, where different set of parameters and use the one which performs better for the specific problem. It is important to note that, since classic RL algorithms (which are widely used in the dialogue domain) are sample-inefficient, they cannot be used to perform (direct) policy optimization using available dialogue corpora.

### 6.5.2 Online/on-policy dialogue optimization

In this section, policy optimization using online/on-policy algorithms is discussed. Experiments focus on optimizing the restaurant information dialogue system. However, since the algorithms are on-policy, the DIPPER dialogue management engine (plugged with KTD-SARSA or SARSA policy learner) is used in conjunction with an n-gram user simulation. In order to cope with the exploration vs exploitation dilemma in case of KTD-SARSA two action selection schemes are used: (i) the standard $\epsilon$-greedy and (ii) a bonus greedy approach.

All three algorithms (*i.e.*, KTD-SARSA $\epsilon$-greedy, KTD-SARSA bonus-greedy and SARSA) uses the same parametrized representation of the $Q$-function. For KTD-SARSA, both prior and variance are set to 1. For KTD-$Q$, the observation noise is set to 1. The learning rate $\alpha$ is set to 0.2. The exploration rate of $\epsilon$-greedy is set to 0.1 and for bonus greedy the $\beta_0$ is set to 1 and the $\beta$ value is set to 5, 3 and 1. Recall from Eq. 6.11 that the $\beta$ parameter of the bonus greedy policy determines to what extent the policy will be explorative with regard to the variance of $\hat{Q}_\theta(s, a)$. Large values for $\beta$ will result in large bonuses. Thus, the policy behaves in an explorative manner during the initial stages of the learning and then tends to act greedily as the variance tends to 0 or in the later stages of learning. The extent of exploration while using KTD-SARSA with bonus-greedy can be observed in Figure 6.4. When $\beta$ is set to 5 the behavior is more explorative when compared to the policy with $\beta$ set to 3 and 1. More exploration results in retrieving a better dialogue policy from fewer number of episodes. Thus at the end of the first 100 training episodes, the average discounted reward obtained with $\beta$ set to 5 is relatively more when compared to the rest.

It may be useful to note that the discounted sum of rewards obtained by the policy learners (KTD-SARSA and SARSA) are averaged using a sliding window (of size 100 initially and 500 later on). The average rewards obtained by SARSA, KTD-SARSA with $\epsilon$-greedy and KTD-SARSA with bonus greedy are shown in Figure 6.5. Experimental results show that KTD-SARSA with bonus greedy learns faster and stabilizes sooner when compared to the rest. KTD-SARSA with $\epsilon$-greedy exploration also learns faster and stabilizes with a better policy when compared to SARSA. Ideally a second order algorithm such as KTD-SARSA is expected to outperform first order algorithms like SARSA. This is due to the fact that, second order algorithms are known to have faster rate of convergence and thus enjoys better sample efficiency. Whereas, SARSA is well known to be sample-inefficient *i.e.*, data intensive and thus normally needs vast amounts of data to retrieve an optimal policy.

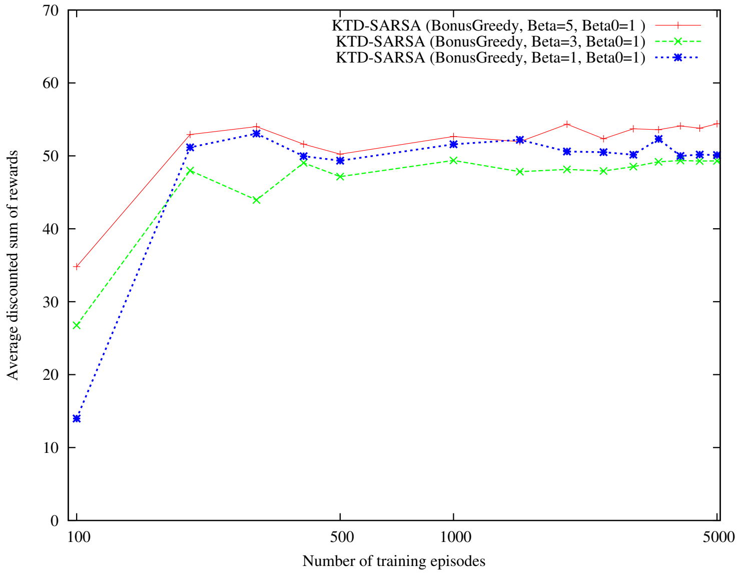


Figure 6.4: *Evaluation of policies learned using KTD-SARSA (Bonus Greedy)*

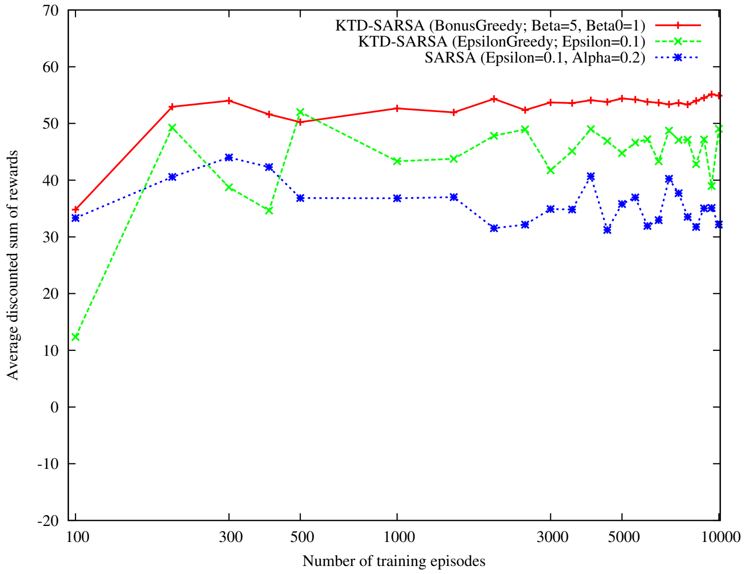


Figure 6.5: *Evaluation of policies learned using KTD-SARSA*

# Part III

# Inverse Reinforcement Learning

# Chapter 7

# User simulation using Inverse Reinforcement Learning

In chapters 5 and 6, sample efficient algorithms for dialogue policy optimization have been discussed. Given the effectiveness showcased by these algorithms, one can now question the basic necessity for employing user simulations in order to perform policy optimization[1]. Using algorithms such as fitted-$Q$ and KTD-$Q$ for direct policy optimization from the available corpus and then improve (if necessary) the resulting policies using online/on-policy or online/off-policy methods such as KTD-SARSA and KTD-$Q$ promises to be an encouraging direction of research on dialogue optimization.

However, one key challenge with regard to policy assessment needs to be addressed. Almost all policy assessment methods proposed so-far in the dialogue domain, either rely on employing user simulators [Eckert *et al.*, 1997a; López-Cózar *et al.*, 2006] or use feedback from experts [Ai & Litman, 2008]. Undoubtedly, evaluating dialogue policies based on feedback from real users (experts or prospective end-users) is the best way forward. However, it is an expensive and time consuming process. Real user feedback on dialogue policies may also be biased by the performance of components such as speech synthesis engine. Even though these modules are inter-related, during evaluation real users are expected to provide feedback only on individual modules. Considering these challenges

[1] Recall user simulators were first introduced to cope with the data requirement of conventional RL algorithms.

and the associated cost factor, a relatively easier option for policy assessment is to continue employing user simulators.

Thus, it is reasonable to mention that user simulators are still required for (at least) evaluating dialogue policies. They continue to be a cost effective and fast means for policy assessment. In the final part of this thesis, focus is shifted towards building robust and adaptive user simulators. There exists several methods for user modelling as discussed in chapter 4. Almost all these methods aim at building user simulators using some reference dialogue corpus. The primary aim here is to ensure statistical consistency between the reference dialogue corpus and the generated synthetic dialogues. Often the dialogue corpora used for this purpose are collected from *several* real users by using Wizard-of-Oz set-up [Rieser, 2008] or using a hand-crafted dialogue manager.

There are several shortcomings with these existing methods for user simulation (recall section 4.3). Firstly, using such a corpus to learn only one simulated user behavior is indeed questionable. The dialogue corpus in itself is collected from multiple users, thus can be expected to showcase multiple behaviors. Summarizing all these behaviors into one (simulated and averaged) user behavior may even result in a behavior which does not correlate with any of the real users or user groups. Such a summarization of a multi-modal distribution by its mean generally results in a very bad estimate. We already know that user simulators have a direct impact on the quality of the dialogue policy [Schatzmann *et al.*, 2005]. Thus, when a user simulation with mean behavior is employed for dialogue optimization, resulting policies are adapted to some non-existing average user. For this reason dialogue corpus should be used to learn multiple user behaviors[1] rather than to simulate some generic or non-existent behavior.

Secondly, real users are known to behave in a goal oriented manner. Some existing user simulation methods do focus on behavior simulation based on user goal like information [Pietquin, 2005; Schatzmann *et al.*, 2007a]. Often, the goal of the user is learned from the data or using ontology of the domain, for example: restaurant information, *etc.* This can be seen as an effort to produce goal-oriented

[1] For instance, methods such as vector quantization which provides ways to approximate a distribution can be used to differentiate distributions which correspond to different user behaviors.

user simulation but they fail to capture the user goal precisely. This is because, firstly the user goal is not observable in the data and secondly user goal may change over time [Mehta *et al.*, 2010]. Real user behavior is not just goal oriented but also adaptive to the other party involved in the communication. In other words, while interacting with any dialogue manager (SDS) real users tend to: (i) interact in their preferred way (*i.e.*, use information rich speech act for faster dialogue or prefer step-by-step system initiated dialogue), (ii) gain expertise in the dialogue domain after several interactions and finally (iii) adapt their behavior (so that they can accomplish their goal).

A novel approach for building user simulators based on Inverse Reinforcement Learning (IRL) is presented in this chapter. This method treats the task of user simulation as a sequential decision making problem and models it as an MDP. Here, at each time-step, decision on user act selection is made based on the dialogue context summarized in the form of the user state. The reward function used to optimize the behavior of the user modelled as an MDP is learned from the data annotated in a user perspective. *It is important to note that we are not learning the goal in terms of task completion but in terms of naturalness of the interaction according to user preferences or expertise.*. In other words, here the focus is on finding a reward function which can reproduce a behavior similar to the observed user behavior and not on determining the user goal at their intention level and try to accomplish it in the shortest possible manner. The following chapter is organized as follows: Equivalence of the task of user simulation to a sequential decision making problem is discussed in Section 7.1. Inverse Reinforcement Learning (IRL) and user simulation based on IRL are elicitated in Section 7.2. Eventually Section 7.3 outlines the experimental set-up and analyse the results. The IRL-based approach proposed here, has the ability to learn multiple simulators when combined with a vector quantization scheme. Each of these simulators reflect different user behaviors that can be observed in the corpora. Users modelled as MDPs can also evolve or co-adapt to the dialogue manager, similar to real users. Behavior specific simulation and co-adaptation are discussed in detail in Chapters 8 and 9 respectively.

## 7.1 User simulation as a sequential decision making problem

Real users while interacting with the dialogue system tend to have an internal goal or objective function. For instance, an user may wish to find information about a moderately priced Italian restaurant in the city centre. The primary aim of users is to obtain this information at the end of the dialogue episode in a preferred manner. At any dialogue turn, users decide what to say next based on the dialogue history and their goal. This essentially involves a sequence of decisions. Thus, in general the task of user simulation can be perceived as a sequential decision making problem (symmetrical to that of the dialogue management task). Being a sequential decision making problem, user simulation can be modelled as an MDP. Let us term the MDP-based user simulation as *User-MDP*. Solving the User-MDP will result in a policy which is expected to behave like real users. It may be useful note that the dynamics of the User-MDP is induced by the dialogue manager with which it interacts. The rest of this section explains how user simulators can be modelled as MDPs and how these MDPs can be solved.

Most existing methods for user simulation focus on reproducing transitions (between dialogue manager and user) or partial trajectories as observed in the reference dialogue corpus. The primary motivation of the work presented here is to reproduce consistent full trajectories. Treating user simulation as a sequential decision making problem will implicitly help us in learning a (simulated) behavior for User-MDP to accomplish the designated goal. The interesting fact here is that apart from accomplishing the goal, the simulated user behavior will be subjective to the dialogue manager in use (*i.e.*, adapt to the other party involved in communication just like real users). Since policies are used for behavior simulation, multiple policies can now be employed to simulate different user behaviors with relative ease.

### 7.1.1 Casting user simulation as an MDP

As discussed so-far (with regard to dialogue management), MDPs provide a formulated way for solving sequential decision making problems. To begin with, the user simulation task has to be casted as an MDP. We know that an MDP is defined as a tuple $\{S, A, P, \gamma, R\}$ (recall Section 3.1). This involves defining the state space ($S$) and the action space ($A$) of the user simulation. Action space of the User-MDP can be defined using the set of all possible user acts for the dialogue problem under study. For instance, in case of restaurant information dialogue system a partial list of user acts includes actions such as: provide-restaurant-type, provide-location, provide-price-range. Let us assume that user behavior optimization is performed in a model-free set-up and thus transition probabilities ($P$) need not be defined. In order to penalize delayed rewards, $\gamma$ can be set to 0.95.

With regard to the state space definition, things get little tricky. In case of a system initiative dialogue strategy, user simulation is expected to respond to the current dialogue act. Thus information regarding the current dialogue act ought to be included in the state representation of the User-MDP. Apart from this, the state representation should also include a summary of all information exchanged since the beginning of the dialogue episode (*i.e.*, dialogue history). This can be achieved by using the Information State paradigm similar to that of dialogue state representation.

For instance, the state representation of the User-MDP for the restaurant information dialogue system will be of the form: $\{dact, \phi_1, \phi_2, \phi_3\}$, where $dact$ is the current dialogue act and $\phi_i$ can take values 0 (which means that the slot information is yet to be furnished to the dialogue manager) and 1 (which means that the slot information has already been furnished to the user). In this case, $\phi_1$ corresponds to restaurant-type, $\phi_2$ corresponds to restaurant-location and $\phi_3$ corresponds to restaurant-price-range. Now that the state space, the action space and the discount factor have all been defined, the only component of the User-MDP yet to be defined is the reward function $R$.

### 7.1.2 User behavior imitation

Let us consider that an user is looking for a moderately priced French restaurant in the northern part of the city. By observing the dialogue trajectory apart from knowing the user preferences for the restaurant (primary goal), other information such as expertness (secondary goal) of the user can also be observed. For instance, expert users may wish to achieve the intended task in fewer number of dialogue turns, whereas novice users may wish to conduct a longer and much informative dialogue. The primary goal outlines what the user wants, and the secondary goal outlines how the user intends to interact with the dialogue system (in order to achieve the primary goal).

In order to perform policy optimization for User-MDP using reinforcement learning, the reward function has to be defined beforehand. The reward function of an MDP provides a compact representation of the task to be achieved by the agent. While employing MDPs for user modelling this reward function should not focus only on summarizing the dialogue task to be performed by the user, rather it should also explain how users prefer to perform some task (elicit their preferences, expertise *etc*). Often the exact reward function of the user is not known. However, some implicit observations of the user goal can be obtained from the dialogue corpus as explained using the above example.

Specifying the reward function for the User-MDP is a critical step in user policy estimation. We know that the reinforcement learning agents merely tend to optimize the agent's behavior according to the specified reward function. Given this fact, one possible option to specify this function, is to make use of the system designer's ability based on his/her observations of users. This option is widely practised in the dialogue domain with regard to reward function specification for MDP-based dialogue managers. However, it is important to note that the task studied is to simulate behavior as observed in dialogue corpus. Thus, guessing the unknown reward function of the real user based on manual observation of a dialogue corpus is not a practical option. Therefore, a more interesting option would involve automated schemes which can be used to retrieve the reward function of the user from the dialogue corpus. One such automated scheme to retrieve the reward function of the sequential decision making agent is the Inverse Reinforcement Learning paradigm [Sutton & Barto, 1998]. In the meantime it is important to note that different users may have different behaviors and hence different utility functions. In chapter 8, means to identify and utilize different utility functions are discussed in detail. For now, our focus is to retrieve a generic reward function from dialogue corpus and utilize it to optimize User-MDP.

## 7.2 Inverse reinforcement learning

Given that the MDP framework is adopted, IRL [Ng & Russell, 2000] aims at learning the reward optimized by an *expert* from a set of observed interactions. The expert is assumed to act optimally with regard to their own reward function. For the sake of user simulation, real users are treated as experts whose behavior ought to be imitated. However, IRL in itself is an ill-posed problem and there exist several possible reward functions that can match the expert behavior [Ng & Russell, 2000]. For instance, any policy is optimal for the null reward function. Thus the primary focus of most existing IRL algorithms is to retrieve some reward function (not necessarily the true reward function) which can result in a behavior similar to that of expert's behavior.

### 7.2.1 IRL: problem elicitation

Let us revisit the definition of User-MDP outlined in Section 7.1.1. In this case, the User-MDP was proposed to take the form $\{S, A, P, \gamma, R\}$. However, as discussed so far the utility function of the real user is unknown. Thus, the User-MDP can now be defined by the tuple $\{S, A, P, \gamma\}/R$, where $/R$ means that the reward function is not available. Let us assume that, the reward function can be (linearly) parametrized using a set of weights $\theta_i$ (where i = 1 ... k) and corresponding set of features $\phi_i$. Then, the reward function $R$ of the User-MDP can be expressed as: $R_\theta(s,a) = \theta^T\phi(s,a) = \sum_{i=1}^{k}\theta_i\phi_i(s,a)$. Recall from section 3.2 that the $Q$-function of an MDP can be defined as:

$$Q^\pi(s,a) = E[\sum_{i=0}^{\infty}\gamma^i R(s_i,a_i)|s_0 = s, a_0 = a] \tag{7.1}$$

Given that the reward function $R$ is now parametrized in terms of $\theta$ and $\phi$, the $Q$-function of the User-MDP can now be defined as:

$$\begin{aligned}
Q^{\pi}(s,a) &= E[\sum_{i=0}^{\infty}\gamma^{i}\theta^{T}\phi(s_i,a_i)|s_0=s,a_0=a];\\
&= \theta^{T}E[\sum_{t=0}^{\infty}\gamma^{t}\phi(s_i,a_i)|s_0=s,a_0=a];\\
Q^{\pi}(s,a) &= \theta^{T}\mu^{\pi}(s_i,a_i). \qquad (7.2)
\end{aligned}$$

where $\mu^{\pi}(s,a)$ is the *feature expectation* of the policy $\pi$. Feature expectation in itself is a vector which contains the discounted measure of features according to state-action visitation frequency (based on when the state is visited in the trajectory). It provides a compact summary of the user behavior observed in the form of trajectories (in dialogue corpus). Let us assume that $m$ dialogue trajectories are made available. Here $H_i$ is the length of the $i^{th}$ dialogue episode from the expert (*i.e.*, real users who act based on some unknown policy $\pi$). Let $(s_t^i, a_t^i)$ be the state-action pairs visited at time-step $t$ in the $i^{th}$ dialogue episode. The feature expectation $\mu^{\pi}(s_0,a)$ can be estimated using Monte Carlo roll-out:

$$\mu^{\pi}(s_0,a) = \frac{1}{m}\sum_{i=0}^{m}\sum_{t=0}^{H_i}\gamma^{t}\phi(s_t^i,a_t^i). \qquad (7.3)$$

The primary focus of IRL is to retrieve some reward function (through its parameters $\theta$). This reward function can in turn be used to predict a simulated user behavior ($\pi_{\text{predict}}$) being similar to the behavior of the expert ($\pi_{\text{expert}}$). However, to ensure the effectiveness of the retrieved reward function, some measure of similarity or dissimilarity between $\pi_{\text{predict}}$ and $\pi_{\text{expert}}$ should be introduced. A majority of IRL algorithms aim at observing minimal dissimilarity ($J$) between behaviors:

$$R_{\theta^*} = \text{argmin}\{J(\pi_{\text{expert}},\pi_{\text{predict}})\}. \qquad (7.4)$$

Here $J$ is some dissimilarity measure between the expert behavior and the predicted user behavior. Recall that from section 3.2, an effective way for comparing two different policies is to compare them using the associated value functions. Comparing two different user behaviors (policies) in terms of their feature expectations (see Eq (7.2)) is indeed comparing the behaviors based on their value function. Assuming that $\|\theta\| \leq 1$ (which is not restrictive: rewards are bounded and scale invariant), one has $\|Q^{\pi}\| \leq \|\mu^{\pi}\|$. As a consequence, an easy way of computing the dissimilarity between the expert behavior $\pi_{\text{expert}}$ and the predicted user behavior $\pi_{\text{predict}}$ is:

$$J(\pi_{\text{expert}}, \pi_{\text{predict}}) = \|\mu_{\text{expert}} - \mu_{\text{predict}}\|^2 \tag{7.5}$$

Notice that from Eq (7.1), a reward function that is non-zero only in some states can lead to a $Q$-function that is non-zero in every state. A greedy policy $\text{argmax}_a \, Q(s, a)$ can be inferred for every state using the retrieved $Q$-function. In simple terms, even if the reward function retrieved from the data is very sparse, it is possible to learn a $Q$-function which has several non-zero elements. This function can then be expected to have non-zero values for expected outcome of many state-action pairs which are originally not seen in the training dialogue corpus. Therefore, IRL-based user simulation can generalize to unseen situations which is harder to obtain from traditional statistical simulation. Also it is useful to note that the primary focus in case of user simulation is to imitate the expert user behavior. Thus the rest of this section outlines an imitation learning algorithm [Abbeel & Ng, 2004] which is uses IRL implicitly to retrieve some $\pi_{\text{predict}}$ that closely correlates with $\pi_{\text{expert}}$ (as shown in Eq. 7.5). However, there exist a class of IRL algorithms (for example: Boularias *et al.* [2011]) which are primarily used to retrieve a reward function. A short overview on such an algorithm and its potential use in the dialogue domain are presented in Chapter 9.

### 7.2.2 Imitation learning algorithm

Apprenticeship learning algorithm proposed in [Abbeel & Ng, 2004] focuses on learning to imitate the behavior (of experts) observed in the sample trajectories. This learning involves two steps in an iterative manner: (i) IRL and (ii) RL. To begin with, IRL is employed to retrieve some underlying reward function from the dialogue corpus. Here a reward function is estimated based on the comparison between the feature expectations of the expert policy and predicted policy. This function is then used by RL or ADP algorithms (such as LSPI or Fitted-$Q$) to

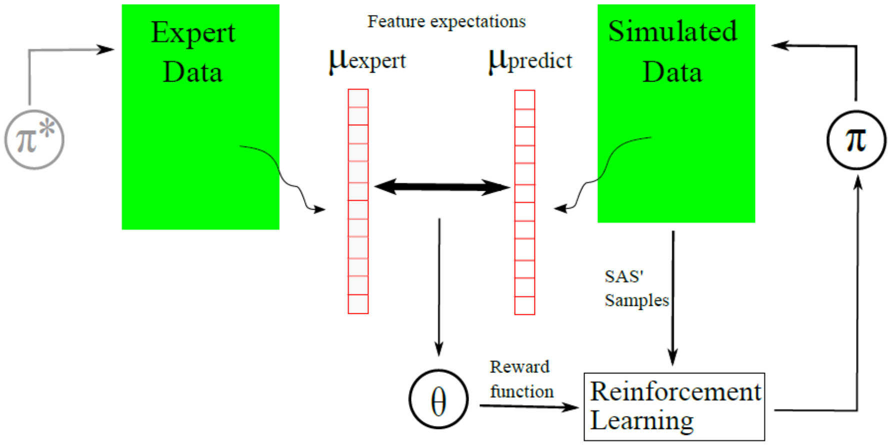


Figure 7.1: *User simulation using imitation learning*

estimate the optimal value function and thus an optimal policy for solving the User-MDP. The resulting policy (since it is estimated using a reward function retrieved from corpus) is expected to imitate or simulate the behavior of real users observed in the dialogue corpus. Imitation learning scheme explained here is pictorially presented in Figure 7.1.

In order to perform imitation learning at every iteration we need two sets of data: (i) Expert data and (ii) Simulated data (see Figure 7.1). Expert data with regard to user simulation will be a set of $m$ dialogue episodes (trajectories) from real users. Feature expectation of expert policy can be computed using the dialogue trajectories as shown in Eq. 7.3. During initialization (*i.e.*, in the first iteration) no policy is available for User-MDP (in order to generate data). Thus, some random policy is used for (simulated) data generation in iteration $j = 1$. However, after the first iteration of the algorithm we will have some predicted policy for User-MDP. Thus, for subsequent iterations $(j+1)$, the policy predicted in iteration $(j)$ is used for dialogue generation[1]. Similar to expert data, simulated data will be a set of $(n)$ dialogues generated using the estimated policy for the User-MDP. Feature expectation can be computed using this data. However, it is important to note that simulated data and its feature expectation changes from one iteration to another. Let us term $\pi_{\text{expert}}$ the policy of the real users and

[1] Data is generated based on the interaction between the User-MDP (with the predicted policy) and the SDS-MDP (dialogue manager with a hand-crafted policy)

$\pi_{\text{predict}}$ the estimated policy of the User-MDP. Feature expectation $\mu_{\text{expert}}$ and $\mu_{\text{predict}}$ can be computed as shown in Eq (7.3 using a Monte Carlo roll-out as explained before). Pseudo code for user simulation using imitation learning is presented in Algo 6.

**Algorithm 6:** User simulation using imitation learning

```
1. Estimate $\mu_{\text{expert}}$ from dialogue corpus
2. Initiate $\Pi$ with random policy $\pi_{\text{predict}} = \pi_0$ and estimate $\mu_{\text{predict}}$
3. Compute $t$ and $\theta$ such that
       $t = \max_{\theta}\{\min_{\pi_{\text{predict}\in\Pi}} \theta^T(\mu_{\text{expert}} - \mu_{\text{predict}})\}$ s.t. $\|\theta\|^2 \leq 1$
4. Terminate if($t \leq \xi$)
5. Optimize $\pi_{\text{predict}}$ for reward $R(s,a) = \theta^T\phi(s,a)$ with RL (LSPI).
6. Compute $\mu_{\text{predict}}$ for $\pi_{\text{predict}}$; $\Pi \longleftarrow \pi_{\text{predict}}$
7. Goto to step 3.
```

Step 3 is an IRL step where based on the dissimilarity between the human and simulated user behavior, a reward function is estimated. This algorithm treats the task of retrieving a prospective reward function as a quadratic programming problem. A reward function (which indeed is a hyperplane in space spanned by the set of features $\phi$) which minimizes the dissimilarity between the two behaviors as well as the one that is maximum apart from the previous retrieved reward functions is estimated in Step 3. In other words, this algorithm searches for the reward function which maximizes the distance between the value of the expert and any policy computed in prior iterations. Apart from estimating a prospective reward function, feature distance $t$ is also computed in Step 3. If $t$ is within some threshold ($\xi$) the algorithm is terminated.

The estimated reward function is then used to perform RL based policy optimization in Step 5. This process results in finding a new policy for the User-MDP. The resulting policy $\pi_j$ in iteration $j$ is added to the set of policies $\Pi$. Synthetic dialogues are generated using the retrieved policy in order to compute the feature expectation ($\mu_{\text{predict}_\text{j}}$). Steps 3 to 6 are performed iteratively until convergence criteria is met. Upon convergence the imitation learning algorithm provides a set of policies $\Pi$. Since these policies are estimated using the predicted reward function they can be expected to imitate the expert behavior. One other interesting fact about these policies is that they can generalize (user behavior for situations unseen in corpus) through the value function which is non-zero for several state-action pairs. In order to simulate the real user (behavior) one can choose the best policy from the set $\Pi$ based on its distance with $\mu_{\text{expert}}$ or choose to use multiple policies with an associated policy selection probability (explained in Section 7.3).

It is important to note that this algorithm does not guarantee to retrieve the true reward function of the real user (which in fact is not possible given the problem is ill-posed). However, the retrieved policies can be expected to imitate the expert precisely. Even though it may seem that using the state-action visitation frequency in the form of feature expectation is comparable to existing approaches for user simulation (such as the n-gram method), it is worth noting that the feature expectation is not directly used. Also feature expectation take a better account (than n-grams) of the dialogue trajectories since the features are discounted (using $\gamma$) based on when they are visited in the trajectory. It is used to predict a reward function which in turn is used to simulate the expert behavior. RL based optimization using the predicted reward function means that the simulated behavior is subjective to the environment the agent is interacting with. For MDP-based user simulation the environment is expected to be a hand-crafted dialogue manager, thus the resulting user behavior is not just aimed at statistically correlating with the reference corpus but behaves like a real user (by adapting its behavior given the observations). However, in order to emulate exactly the same behavior of the expert, the algorithm discussed here expects to have an environment as close as possible to the one that was originally used for expert interaction. Most existing ML-based approaches for user simulation focus on simulating the user at the transition level when the primary focus of this method is to simulate user trajectories. Casting the task of user simulation as an MDP is precisely meant to simulate real user like trajectories in an adaptive manner.

## 7.3 Experimental results and analysis

In this section, a simple experiment is set-up to showcase the user simulation ability of the proposed IRL based method. Restaurant information dialogue system considered so far is reused for this purpose. However, unlike previous chapters (which focused on dialogue management), here the focus is on building a user simulation for the restaurant information dialogue task. To recall, the aim of the dialogue system is to give information about restaurants in a city based on specific user preferences. In order to employ imitation learning algorithm for user simulation, feature expectations ought to be computed. This computation requires dialogue trajectories from both experts (real users) and as well as predicted policies (Monte Carlo roll-out). Dialogue corpus generated using real users can be used to compute $\mu_{\text{expert}}$. However, synthetic dialogues have to be generated using a hand-crafted dialogue manager and a user simulator controlled by the estimated policy $\pi_{\text{predict}}$, in order to compute $\mu_{\text{predict}}$. It may be useful to recollect that the dynamics of the User-MDP is induced by the SDS-MDP. For this reason during corpora generation (expert trajectories) as well as during imitation learning the same SDS-MDP is employed.

The task of dialogue management for the restaurant information system is casted as an MDP. Let us term this as SDS-MDP. The dialogue state of SDS-MDP is composed of knowledge corresponding to 3 slots: location, cuisine and price-range. Thus a list of possible system (dialogue) acts includes 14 actions: Ask-slot (3 actions), Explicit-confirm (3 actions), Implicit-confirm and Ask-slot value (6 actions), Close-dialogue and Close-Session. Setting on the task to optimize the SDS-MDP using the available dialogue corpus would be the ideal way to begin the experiment. Since the focus here is on user simulation, a hand-crafted dialogue strategy is employed (for SDS-MDP) to fill and confirm all the 3 slots one after the other and eventually conclude the dialogue session.

As explained in section 7.1.1, the task of user simulation is also casted as an MDP (User-MDP). The user state presents a compact summary of the dialogue course from the user simulation's perspective. For the restaurant information dialogue problem the state of User-MDP has the following representation: $\{dact, \phi_1, \phi_2, \phi_3\}$, where the *dact* field takes values in $\{0, \ldots, 12\}$ representing the

Table 7.1: *Hand-crafted user behavior*

| SystemAct | UserActs (probability) |
|---|---|
| Greet | Silent (0.7) AllSlots (0.3) |
| AskSlot | OneSlot (0.95) AllSlots (0.05) |
| Explicit-Conf | Confirm (1.0) |
| Implicit-Conf | OneSlot (0.9) Negate (0.1) |
| CloseDialogue | Silent (1.0) |

corresponding system acts (action space) of SDS-MDP. Fields $\phi_1, \phi_2, \phi_3$ take values in $\{0, \ldots, 2\}$ with (0) the slot is empty (never provided by the user), with (1) the slot has been provided by the user and with (2) the slot is confirmed. The action space of User-MDP includes the following 10 user acts: remain silent (Silent), provide-all-values (AllSlots), provide-one-value (OneSlot: 3 actions), confirm slot value (Confirm: 3 actions), negate slot value (Negate: 3 actions) and hangup (CloseDialogue). The state-action space of the User-MDP is spanned by a set of 3780 features (*i.e.*, $10.(14.3^3)$). The discount factor of the User-MDP is set to $\gamma$ = 0.95.

### 7.3.1 Learning to imitate

For the sake of simplicity, expert data is generated using a hand-crafted (expert) user. The proposed user behavior for imitation is detailed in Table 7.1. A set of 1000 dialogue trajectories is generated using this user and the SDS-MDP. Feature expectation of the expert ($\pi_{\text{expert}}$) is computed using the generated dialogue trajectories as shown in Eq. 7.3. Ideally speaking, during initialization $\pi_0$ is not available to generate data (see figure 7.1). Thus, a random (uniform) policy is used to generate this data at $j = 1$. In subsequent iterations $(j + 1)$, the estimated policy $\pi_{\text{predict}_\text{j}}$ is used for generating simulated data. Following which feature expectation ($\mu_{\text{predict}_\text{j}}$) can be computed from these trajectories.

At each iteration, the estimated reward function is used to optimize User-MDP by employing LSPI [Lagoudakis & Parr, 2003][1]. It is important to note that LSPI is a batch algorithm which allows learning from a fixed data set. Upon

[1] Any sample efficient algorithm (as discussed in Part II) can be used for this purpose. During the experiments an incremental version of LSPI [Geramifard *et al.*, 2006] is used (which does not require matrix inversion or parameter tuning).

convergence (here $\xi$ is set to 0.1) the best policy $\pi^*$ for the User-MDP (chosen from $\Pi$ based on the least dissimilarity measure between $\pi_{\text{predict}}$ and $\pi_{\text{expert}}$) will simulate the expert behavior. An example dialogue episode between the hand-crafted SDS-MDP and the User-MDP controlled by policy learned using imitation learning is presented as follows. Here it can be observed that the User-MDP policy

```
=======================================================
  User State: GreetUser 0 0 0
    User Act: Say-Nothing
  User State: Ask-Slot-2 0 0 0
    User Act: Provide-Slot-2
  User State: Implicit-Confirm-Slot-2-Ask-Slot-1 0 1 0
    User Act: Provide-Slot-1
  User State: Implicit-Confirm-Slot-1-Ask-Slot-3 1 2 0
    User Act: Provide-Slot-3
  User State: Confirm-Slot-3 2 2 1
    User Act: Confirm-Slot-3
  User State: CloseDia 2 2 2
    User Act: HangUp %
=======================================================
```

*An example dialogue between the hand-crafted SDS-MDP and IRL-User-MDP*

### 7.3.2 Evaluation of user behavior

It may be useful to note that the behavior of the expert is stochastic (as shown in Table 7.1). However, the behavior of IRL user simulation is deterministic since only one of the policies estimated during training is used during evaluation. One possibility to obtain a stochastic user behavior is to employ a *policy mixer* to pick up a different policy before starting each dialogue episode. However, this mixer has to be trained so that efficient policies are chosen more often. Let the probability of choosing a policy $\pi^i_{\text{predict}} \in \Pi$ be $\lambda_i$. The values of $\lambda_{1\ldots n}$ can be heuristically determined using a Gibbs distribution:

$$\lambda_i = e(-d_i/\tau)/\sum_{j=1}^{n} e(-d_j/\tau). \tag{7.6}$$

where $d_i$ is the distance between $\mu_{\text{expert}}$ and $\mu^i_{\text{predict}}$ estimated during training and $\tau$ is a temperature parameter. During the evaluation, $\tau$ is set to 0.01 (ensures that the policies closest to $\pi_{\text{expert}}$ are given more preference). Let the behavior of IRL user simulation which employs policy mixer be termed as MixIRL-UserSim.

In order to evaluate the performance of the IRL-based user simulation, 3000 dialogue trajectories are generated using SDS-MDP and User-MDP. Of these 3000 episodes: (i) 1000 trajectories are generated by employing User-MDP with the (hand-crafted) expert policy, (ii) 1000 trajectories are generated using User-MDP with a policy $\pi$. This policy $\pi \in \Pi$ is hand picked based on its performance and (iii) 1000 episodes are generated using User-MDP whose policy is selected by a mixer. Figure 8.4 showcases the average choice of user actions per episode by the expert user, IRL user and IRL user which employed policy mixer. It can be observed that the expert user and the mixed IRL user behaviors correlate better (since both of them are stochastic) when compared to deterministic IRL user behavior (which follows one deterministic control policy). In this case, the IRL user exhibits the most commonly or frequently observed expert user.

A similarity measure suggested in [Pietquin & Hastie, 2011; Schatzmann *et al.*, 2005] is also used to compare the behaviors of the three different user simulation. This metric is used to measure the correlation between the trajectories of the expert, $\pi_{hc}$ and the IRL user model, $\pi_{irl}$:

$$Sim(\pi_{\text{expert}}, \pi_{\text{predict}}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{1+\mathrm{rank(a_i)}}$$

with $n$ the number of user acts observed in the trajectory generated by the expert user model. Ranking of user acts (selected by $\pi_{\text{expert}}$) based on their $Q$-value (obtained using $Q$-function of the IRL used model) gives an indication to what extent the user acts selected by the expert model correlates with the learned IRL user model. Upon comparing the expert user model with the IRL user model,

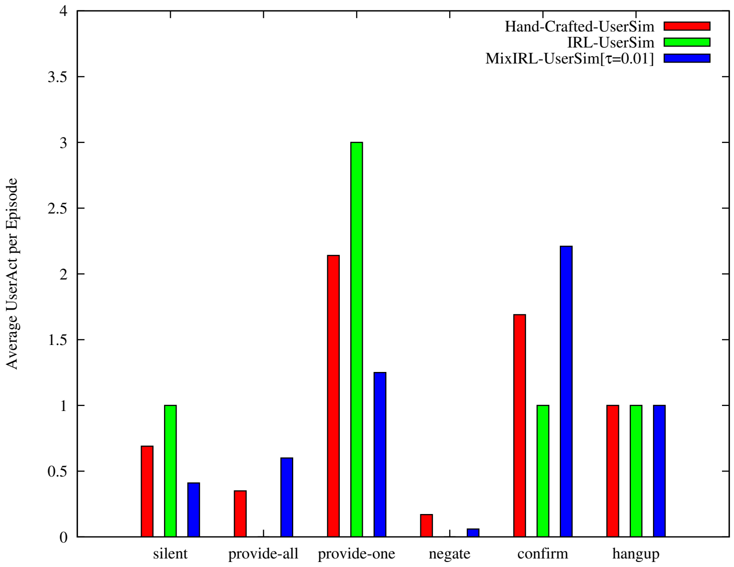


Figure 7.2: *Frequency of user actions per episode*

the similarity measure is found to be 0.48 for IRL-UserSim and 0.46 for MixIRL-UserSim which are close to the optimal value 0.5 (both rankings are identical, and then rank(.) is 1 constantly). This confirms that the behavior of the IRL user model correlates very well with the expert's behavior.

### 7.3.3 IRL evaluation metric

The primary motivation behind IRL user simulation is to perceive the task of user simulation as an MDP and optimize it using the predicted reward function. Even though the predicted reward function is not guaranteed to be the true reward function of the real user, it is one which results in a similar behavior. Thus, it provides a compact representation of user's intentions and preferences. This function can in turn be used to quantify the performance of any given user simulation. Let us call this new metric for evaluating user simulations as *IRL evaluation metric*. Table 7.2 shows the average discounted reward obtained by the user simulations based on the estimated reward function. It can be observed

Table 7.2: *Hand-crafted vs IRL user behavior*

| UserSim | AvgLength | AvgReward |
|---|---|---|
| Hand-crafted | 6.03 | 2.6 |
| IRL | 6.0 | 2.7 |
| Mixed-IRL | 5.5 | 2.9 |

that the rewards obtained by the IRL user simulations and the expert are indeed similar. Also the average lengths of dialogue episodes, both in case of expert and simulated user, are almost the same. These evaluation metrics ascertain the hypothesis of similar behaviors (which originally was the intention of the experiment).

# Chapter 8

# User behavior clustering

Corpora generation is an expensive and time consuming task in the dialogue domain since it involves real users. These users come from different backgrounds and are thus expected to showcase a wide range of user behaviors. For instance, expert users may wish to achieve the intended task in fewer number of dialogue turns, whereas novice users may need to conduct a longer and much informative dialogue. Since dialogue corpora is generated from multiple users, our aim during user modelling should be on identifying and learning all possible (different) user behaviors rather than simulating one generic behavior.

First step in building behavior specific user simulators is to annotate the dialogue corpus from a user simulation perspective. Once this task is accomplished, existence of different user behaviors in the dialogue corpus has to be ascertained. Then efforts should be made to analyse how these behaviors differ from each other. Most often dialogue trajectories collected from real users are annotated with human intervention. Thus one possibility would be to identify and differentiate user behaviors using manual analysis. This would not only increase the cost of corpora generation and annotation process, but also is less effective and exponentially (with increase in problem complexity) time consuming process. Thus automated schemes should be employed to differentiate user behaviors observed in the dialogue corpus.

The key challenge in employing such an automated scheme is the fact that dialogue episodes (trajectories) vary widely based on the dialogue context/domain and the communicative goal. Thus directly comparing dialogues is not feasible.

This chapter presents an automatic scheme for clustering behaviors of dialogue system users based on expected discounted cumulative feature vectors. The following chapter is organized as follows: to begin with, the proposed scheme for clustering user behaviors is outlined in Section 8.1. Once similar behaviors are identified, specific behaviors can then be imitated using IRL-based user simulation (as shown in Sections 8.1.1 and 8.1.2). The process of building behavior specific user simulators is discussed in Section 8.1.3. Eventually Section 8.2 outlines a set of experiments carried out to validate the effectiveness of the proposed methods.

## 8.1 Quantizing and clustering trajectories

Identifying and simulating different user behaviors from dialogue corpus is a crucial task in user modelling. For instance, it is important to simulate all observed behaviors in order to estimate truly user adaptive dialogue strategies. However, until now, dialogue corpus is annotated in dialogue management perspective[1]. This provided little scope to observe different user behaviors (only user actions can be used to observe user behaviors).

### 8.1.1 Modelling users with MDPs

In Chapter 7 the task of user simulation has been treated as a sequential decision making problem and casted as an MDP. The primary motivation for this work comes from the fact that a real user can be perceived as a decision maker optimizing some unknown reward function. For this purpose dialogue corpus is annotated in a user simulation perspective (capturing user state transitions). Such an annotation scheme provides rich information to identify and analyse different user behaviors. However, dialogue trajectories can hardly be used directly, notably because they tend to vary widely in length (*i.e.*, number of dialogue turns). Let us revisit the restaurant information dialogue system. User simulation of this SDS can be casted as an MDP as discussed in Section 7.1. Once the User-MDP

[1] Annotation of trajectories primarily intends to capture all dialogue state transitions.

is formulated, a policy for imitating real user behavior observed in the dialogue corpus can be learned.

### 8.1.2 Discounted cumulative feature vectors

The immediate effect of treating user simulation as an MDP results in perceiving each user as a policy for User-MDP. The initial state $s_0$ is always the same: there is no system act and all slots are empty. Let us assume that the dialogue corpus is a contribution of $K$ different users or groups of users, and therefore $K$ different policies $\pi_k$. Recall that the IRL-based user simulation summarizes user behavior in the form of feature expectations (see Eq. 7.3). This in fact is the (averaged) sum of the so-called expected discounted cumulative feature vectors (explained as follows). Let us assume that the dialogue corpus contains $N$ dialogues, each one of length $H_n$. For each of these trajectories, we can compute the corresponding discounted cumulative feature vectors as follows:

$$\delta_n = \sum_{t=0}^{H_n} \gamma^t \phi(s_t). \tag{8.1}$$

Most interesting fact about these vectors is that they provide a compact summary of dialogue episodes. Until recently [Chandramohan *et al.*, 2011a] such a compact representation was not introduced to the dialogue domain. Since these vectors are not subjective to dialogue length they can now be used to compare trajectories. To begin with, this measure is used for clustering real users observed in the dialogue corpus based on their behaviors.

Each of these $\delta_n$ vectors succinctly represents a dialogue trajectory. It provides an unbiased estimate of $\mu_{\pi_k}$ for some users (represented by the policy $\pi_k$). While clustering, it is assumed that the intra-variability of one user behavior (due to randomness of transitions) is lower than the inter-variability between user groups. Being a vector in itself, $\delta_n$ exists in some high dimensional space $\mathcal{R}^p$, where $p$ is the number of features (recall Section 7.2). Thus, any vector quantization algorithm can be employed to cluster $\delta_n$ in $\mathcal{R}^p$. Even simple clustering algorithms such as the K-means algorithm can be employed for clustering the discounted cumulative feature vectors. Such schemes will result in clustering user behaviors observed in

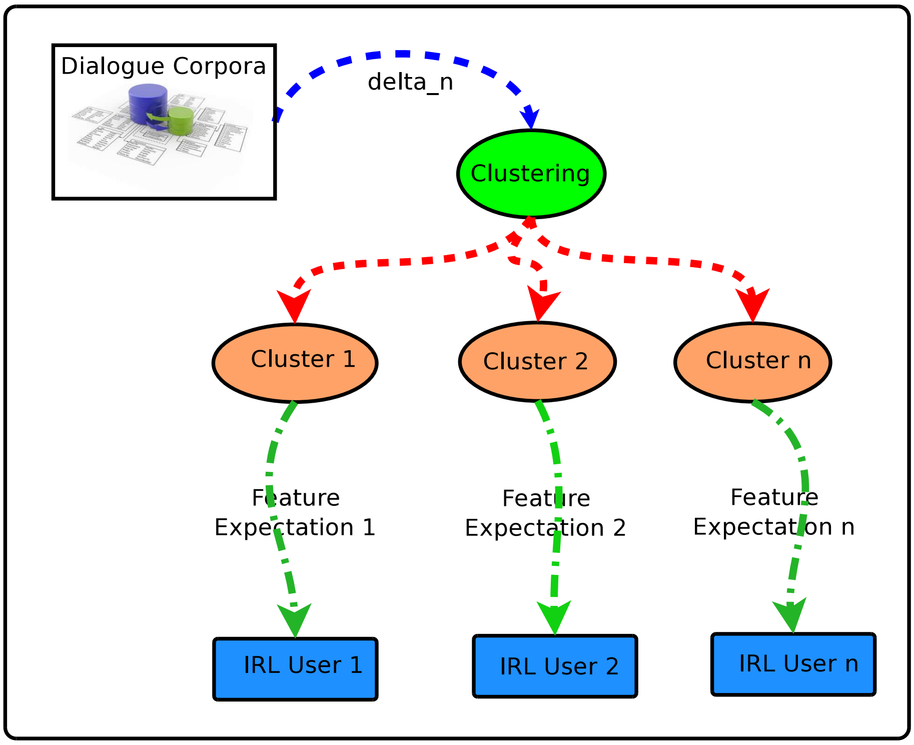


Figure 8.1: *Behavior specific user simulations*

the dialogue corpus.

### 8.1.3 Behavior specific user simulation

The task of building behavior specific (multiple) user simulators can be achieved by combing the clustering scheme with IRL-based user simulation method (or using other methods), as shown in Figure 8.1. Often, data-centric user simulation methods use the dialogue corpus as it is without any pre-processing. This is one of the primary reasons for learning a generic behavior for simulated users. The clustering method proposed in this chapter can be used to identify and differentiate user behaviors. Thus it provides an opportunity to precisely simulate these different behaviors.

In the pre-processing stage, user behaviors observed in the dialogue corpus are automatically clustered based on their $\delta_n$. The basic assumption here is that each of the resulting clusters have unique behavior. In order to build behavior specific simulators, feature expectation is computed only using dialogue trajectories included in one of the resulting clusters. Performing IRL-based user simulation using such a feature expectation will result in imitating the behavior observed in the individual clusters. Upon repeating the process for each cluster, it is theoretically possible to build multiple user simulators with different behaviors.

## 8.2 Experimental results and analysis

In this section, a series of simple experiments is performed to validate the effectiveness of the proposed methods for user clustering and behavior specific user simulation. To begin with, clustering is performed on dialogue trajectories obtained for the 3-slot restaurant information problem. Here, two different hand-crafted users are employed for data generation. This is done for the sake of simplicity and also to ensure the presence of (at least) two different user behaviors in dialogue corpus. Following this, clustering is performed on dialogue trajectories (generated by real users) for the 12-slot Cambridge town information dialogue system [Keizer *et al.*, 2010]. The experiment with 12 slot dialogue systems is proposed in order to determine the effectiveness of the proposed method on problems for which the number of underlying clusters is not known beforehand. Finally behavior specific user simulators are built for the 3-slot restaurant information dialogue problem.

### 8.2.1 Behavior clustering for 3-slot dialogue problem

The primary goal of this experiment is to explore the possibility of user clustering based on discounted cumulative feature vectors. The state representation of SDS-MDP and User-MDP discussed in section 7.3 are reused for the experiments. Two simple hand-crafted user policies for the User-MDP are specified as shown in Table 8.1. User behavior 1 outlines a proactive user who prefers to furnish *all* the required information as soon as the system greets. Whereas user behavior 2 outlines a novice user who prefers to furnish *only* the information requested by the system. These user behaviors are indeed chosen to be simple so that the

| SystemAct | UserActs 1 (probab.) | UserActs 2 (probab.) |
|---|---|---|
| Greet | Silent (0.1) AllSlots (0.9) | Silent (0.9) AllSlots (0.1) |
| AskSlot | OneSlot (0.95) AllSlots (0.05) | OneSlot (0.95) AllSlots (0.05) |
| Explicit-Conf | Confirm (1.0) | Confirm (1.0) |
| Implicit-Conf | OneSlot (0.9) Negate (0.1) | OneSlot (0.9) Negate (0.1) |
| CloseDialogue | Silent (1.0) | Silent (1.0) |

Table 8.1: *Hand-crafted users behaviors*

clustering results can be visually interpreted. The SDS-MDP is controlled by a hand-crafted policy that can respond to both user behaviors.

A total of 3000 dialogue trajectories are generated by randomly choosing one of the two user behaviors. The resulting dialogue corpus consists of 1544 trajectories from the expert user model and 1456 trajectories from the novice user model. Discounted cumulative feature vectors corresponding to each of these 3000 trajectories are computed. Following which behavior clustering is performed using K-means method with K set to 2 (since it is known that only two user behaviors are simulated). Euclidean distance measure is used as part of the K means method.

Upon clustering it is observed that the two clusters have 1549 and 1451 elements. Average user action selection per episode is used as the measure to compare the two hand-crafted behaviors with the behaviors observed in the clusters. This measure is computed from the trajectories in the dialogue corpus, cluster 1 and cluster 2. Figure 8.4 presents a histogram of the action selection frequency. It can be observed that the expert user behavior correlates with cluster 1 and the novice user behavior correlates with cluster 2. Since the clustered user behaviors correlate with the respective hand-crafted user behaviors, clustering user behavior based on discounted cumulative feature vector can be deemed to be a viable option.

### 8.2.2 Behavior clustering for 12-slot dialogue problem

For the second experiment, a 12-slot restaurant information dialogue system is considered (same as in [Keizer *et al.*, 2010]). The data used for clustering is gen-

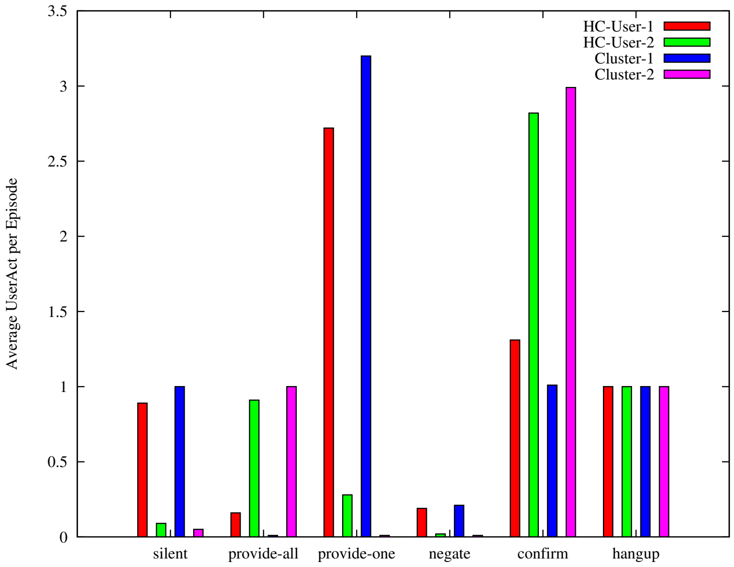


Figure 8.2: *Frequency of user actions per episode for the 3-slot problem*

erated from interaction between a hand-crafted dialogue policy and human users. This problem is a large real world dialogue problem similar to the 3-slot problem, but also includes several other user preferences such as type of drinks served, type of music played, number of stars, *etc.* This experiment primarily aims at determining whether the proposed clustering scheme based on feature expectations: (i) will work on data collected from real users, (ii) whether the method scales for large state space dialogue problems and (iii) whether the resulting clusters are indeed distinct from each other.

User simulation is again modelled as an MDP. The state of the User-MDP is: {System-Act} {Preference} {Goal} {Annoyance} {Correctness} {ChangeIntention}, where system-act can takes values between 0 and 10, preference and goal fields can take values from 0 to 2 (0 means no information is furnished or obtained, 1 meaning partial exchange and 2 means all constraints have been informed or all requested information have been obtained from the system), while annoyance is a boolean value which indicates whether the user is annoyed or not, correctness field indicates whether the information presented by the system is what user actually conveyed and changeIntention field indicates whether the system can find results for the user constraints or not.

The dialogue corpus consists of 480 dialogue trajectories (weakly annotated from user perspective) generated from interactions between human users and a hand-crafted dialogue manager. The associated discounted cumulative feature vectors of all these trajectories are computed. Assuming that these vectors are linearly separable, the K-means method is applied for clustering. The primary challenge is to determine the number of different types of user behaviors. This indeed determines the number of clusters ($K$). However, this information is not known unlike earlier experiment. A possible option for determining the number of clusters is to observe the average cumulative distortion with varying number of clusters. In addition to using average distortion, a measure of intra-cluster cosine similarity (*i.e.*, normalized dot product between two discounted cumulative feature vectors) is also used to decide the number of clusters: $\cos(A, B) = \frac{A.B}{\|A\|.\|B\|}$.

Figure 8.3 presents a comparison of the average distortion and cosine similarity among clusters for several values of K. It can be observed that for K $>$ 4, the distortion as well as the intra-cluster cohesion do not improve significantly. Thus, for experimental purpose K is set to 4. As it can be observed from the User-MDP state, the data used for clustering is symbolic, hence K-means clustering is performed using the cosine similarity measure rather than the Euclidean distance. The 480 trajectories are grouped as 4 clusters each with 35, 124, 114 and 207 elements.

The primary problem while dealing with human user data, unlike the simple simulated user behavior (like the 3-slot problem), is that the behaviors grouped under various clusters may not be easily differentiated. Thus the use of Kullback-Leibler (KL) divergence measure [Kullback & Leibler, 1951] to ensure the correctness of clustering is proposed. The KL divergence, which measures the dissimilarity between two probability distributions $P$ and $Q$ can be defined as:

$$D_{KL}(P||Q) = \sum_{i=1}^{M} p_i log(\frac{p_i}{q_i}) \tag{8.2}$$

where $p_i$ (resp. $q_i$) is the frequency of dialogue act $a_i$ in the histogram of distribution $P$ (resp. $Q$). KL divergence close to 0 corresponds to identical behavior

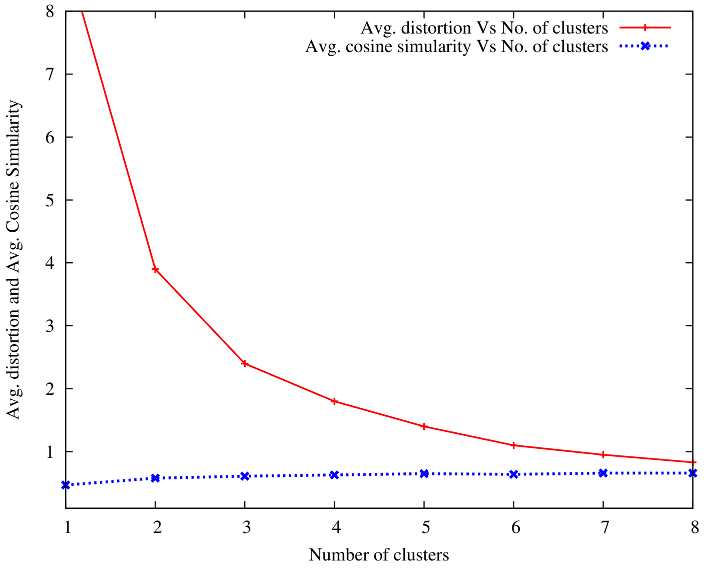


Figure 8.3: *Cumulative distortion with varying number of clusters*

| Behavior comparison | KL-Divergence |
|---|---|
| Cluster-1 vs Cluster-2 | 2.52 |
| Cluster-1 vs Cluster-3 | 2.51 |
| Cluster-1 vs Cluster-4 | 1.99 |

Table 8.2: *Inter-cluster Kullback-Leibler divergence*

and larger non zero values correspond to significant divergence from each other. Having this in mind, if the behavior of the users in two different clusters are truly distinct then there should be a noticeable divergence between the frequency of user acts selection per episode. KL divergence scores are computed using user act selection frequency across all clusters. As it can be observed from Table 8.2 clusters formed by K-means method and their respective user behaviors have significant KL divergence when compared among themselves. This ascertains that every cluster indeed represents a specific behavior (possible group of users with similar behavior) which is different from other clusters.

It is also important to establish that there is a good cohesion among the elements within a cluster, *i.e.*, to show that the intra-cluster cohesion is non neg-

Table 8.3: *Intra-cluster cosine similarity*

| Behavior-Type | Cosine similarity |
|---|---|
| Cluster-1 | 0.68 |
| Cluster-2 | 0.60 |
| Cluster-3 | 0.55 |
| Cluster-4 | 0.70 |

ligible. In order to measure the intra-cluster cohesion, cosine similarity (between the centroid of the cluster and the elements within the cluster) is computed. The value for cosine similarity while comparing 2 elements can range form -1 (meaning totally opposite elements), 0 (meaning cosine independence of elements), 1 (meaning cosine similarity of elements). Cosine similarity is computed for each cluster by measuring the similarity between the cluster's centroid and its own elements. Without clustering, the cosine similarity of the complete set of data to the average vector is 0.4. It can be observed from Table 8.3 that the cosine similarity of all the clusters have values closer to 1. This ascertains similarity between trajectories in each of the 4 clusters.

### 8.2.3 Behavior specific user simulation results

Recall that the resulting clusters for the 3-slot problem have 1549 (Cluster-1) and 1451 (Cluster-2) elements respectively. Normalized sum of $\delta_u$, where $u = 1...1549$ from Cluster-1 yields the feature expectation of the expert user model. Likewise sum of $\delta_v$, where $v = 1...1451$ from Cluster-2 yields the feature expectation of the novice user model. When these feature expectations are fed to the IRL-based user simulation method (one after the other), two different policies for User-MDP can be estimated. Let us term these policies and the resulting behaviors as IRL-User-1 and IRL-User-2. Using these two IRL user models and the hand-crafted SDS-MDP, a set of 2000 dialogue episodes were generated (1000 with each IRL user). User act selection frequency of these IRL user simulators along with the hand-crafted and clustered behaviors are presented in Figure 8.4. From the histogram a strong correlation between the behaviors of the expert user model, Cluster-1

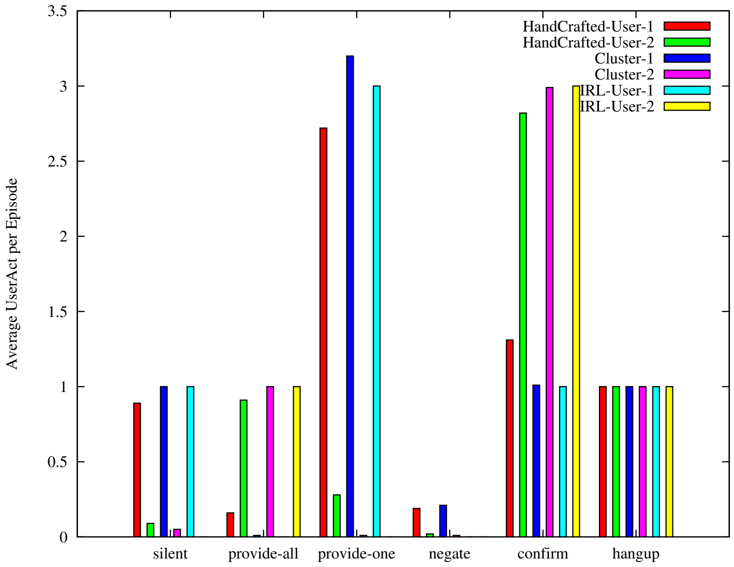


Figure 8.4: *Behavior specific user simulation*

and IRL-User-1 can be observed. Same is the case with regard to the novice user model, Cluster-2 and IRL-User-2. These results indicate the fact that: by combining automatic clustering schemes with IRL-based user simulation methods it is now possible to simulate specific behaviors observed in the dialogue corpus.

The primary advantage here is that now all different simulated user behaviors can be used for dialogue policy optimization as well as for policy assessment. These user simulators can be either employed separately or collectively (mixture of behaviors). The impact of such a scheme on the resulting dialogue policies ought to be studied in the near future. Most certainly multiple user simulators will yield multiple dialogue policies which are better adapted to the specific group of users. The key challenge in such a scenario would be to determine how to switch between policies while in the process of interacting with a user? Even if we find a way to switch between policies, its effectiveness compared to one adaptive policy ought to be ascertained in the near future.

# Chapter 9

# Co-adaptation in dialogue systems

In case of human-human interaction, the parties involved in conversation tend to evolve mutually over a period of time. Most often, when the conversation begins, the parties asses each others ability to understand and then continue to interact based on their initial assessment. This aspect of human-human communication can be termed as *dialogue-initiation*. However, once the dialogue progresses humans tend to evolve mutually (based on the history of conversation). Let this stage of human-human communication be termed as *dialogue-evolution*. An example for such an evolution leads to the use of an acronym MDP (during subsequent references) after defining the Markov Decision Process in oral or written communication. It may be useful to note that the dialogue-evolution occurs at several levels (such as: the amount of information exchanged, terminologies used during the conversation, *etc.*). Naturalness of human-human dialogues can be attributed to several unique abilities of humans (such as: mutual evolution during the period of interaction).

Most existing approaches for dialogue management optimization [Rieser & Lemon, 2011] focuses on retrieving some optimal (with respect to the reward function) and user adaptive (with respect to user simulation) dialogue policy [Eckert *et al.*, 1997b]. This can be perceived as the dialogue-initiation stage in man-machine interaction. This aspect of man-machine interaction has been studied in detail and is now state of the art. However, very less attention has been paid to dialogue-evolution in man-machine interaction. One of the most relevant work done towards dialogue-evolution is to perform on-line policy optimization [Daubigney *et al.*, 2011]. There are two primary drawbacks: (i) when the dialogue manager tries to evolve or optimize its behavior, human users also tend to adapt to the dialogue manager (for example, instead of speaking normally, users provide only information requested by the dialogue manager). These contradicting efforts when applied simultaneously may often result in sub-optimal policies and thus blocking the possibility for dialogue-evolution, (ii) even if dialogue-evolution occurs it may bias the dialogue management to act over confidently [Daubigney *et al.*, 2011; Gasic *et al.*, 2011] and thereby resulting in inferior policies. Changes made directly to the policy used for dialogue management cannot always be guaranteed to improve the performance. In the worst case scenario this may induce very bad user experience. Also, in case of online optimization, the speed of adaptation is relatively slow considering the fact that users (and hence behaviors) encountered are random in nature.

In order to cope with this problem, a novel approach for evolution in spoken dialogue systems is proposed in this chapter. Here, the dialogue manager and the user simulation modelled as MDPs are optimized iteratively over a period of time. Most important advantage of the proposed approach is that: it is now possible to learn optimal policies which are originally not seen in the dialogue corpus. Evolution of the environment helps the RL agent (user/dialogue manager) to estimate better policies during subsequent optimization. This chapter is organized as follows: Some cognitive aspects of human-human communication are discussed in section 9.1. The proposed scheme for co-adaptation in dialogue systems is discussed in section 9.2. Finally section 9.3 describes the experimental set-up and analyse the consequence of co-adaptation for a 2-slot restaurant information SDS.

## 9.1 Cognitive aspects of interpersonal interaction

As suggested by the theory of uncertainty reduction [Baxter & Braithwaite, 2008; Berger, 1986], we humans tend to have difficulties with regard to uncertainty (irrespective of its form) and so we try reduce it by seeking more information. During this process, we constantly evolve over a period of time. For instance, interpersonal relationship [Altman & Taylor, 1973] involves various stages such as orientation (introductory stage), exploratory affective stage (explorative stage), affective exchange (acquaintance stage) and stable stage. At each of these stages people involved try to seek information about each other so that they can reduce uncertainty about each other.

The uncertainty theory also holds good in terms of human-human communication. Here, apart from trying to reduce uncertainty, we also try to follow a set of rules during communication. These rules ascertain that the parties involved in communication interpret the message in an identical manner (which is critical for understanding each other). However, if some previously unknown information is encountered during communication, the associated uncertainty ought to be reduced. The most obvious way this can be achieved is to seek information about the same. However, once the knowledge is updated the associated uncertainty reduces and thus the communication evolve over a period of time. It can be observed from all these cases that, given an opportunity, humans can learn and evolve over time. Ideally speaking, in order to make man-machine dialogues as natural as possible, machines should have the ability to evolve. This evolution can occur at the word level, intention level or task level [Glass & Seneff, 2003]. For instance, a dialogue system's ability to learn new names was showcased in [Chung *et al.*, 2003].

## 9.2 Co-adaptation process elicitation

Co-adaptation in dialogue systems can provide an opportunity for the dialogue manager and user simulation to evolve mutually. To begin with, the task of dialogue management (SDS-MDP) and user simulation (User-MDP) are casted as MDPs. In order to perform policy optimization, reward functions for SDS-MDP and User-MDP have to be defined in advance. As a pre-processing step for co-adaptation, these reward functions can be learned from the dialogue corpus using IRL. It may be useful to recall that in Chapter 7, an algorithm to perform imitation learning by means of IRL was outlined and then employed for the purpose of user simulation. At each iteration, this algorithm requires Monte Carlo roll-out of the estimated policy $\pi_{predict}$ in order to compute $\mu^{\pi}(s_0, a)$. It focuses on estimating a set of policies policy $\pi^{j}_{predict} \in \Pi$ but falls short of retrieving the true reward function. In order to avoid the necessity of Monte Carlo roll-out, in this chapter we focus on using the relative entropy IRL algorithm [Boularias *et al.*, 2011] for experimental purposes. The later algorithm focuses on minimizing the relative entropy between the empirical distribution of the trajectories from a uniform policy and the empirical distribution of trajectories from a expert policy. This minimization is carried out by means of (importance) sample-based gradient.

Unlike dialogue optimization shown in Figure 9.1, co-adaptation is an iterative procedure where policy optimization of User-MDP and SDS-MDP is performed repeatedly. As shown in Figure 9.2, in Step 1: policy optimization for User-MDP is performed using a hand-crafted dialogue manager and the available dialogue corpus. Using the policy learned in Step 1; dialogue optimization for SDS-MDP is performed in the next step. Following which policy optimization for User-MDP is performed using the optimal dialogue policy retrieved in the previous step. Step N and Step N+1 are repeated iteratively until convergence (in other words until the resulting policies cease to evolve).

Even though the resulting policies are deterministic, some amount of stochasticity can be introduced using Gibbs sampling based on state-action values $Q$(s,a). Let the probability of choosing a dialogue act or user act $a_{\mathrm{i}} \in A$ be $\lambda_i$ such that $\sum_{i=1}^{n} \lambda_i = 1$. The values of $\lambda_{1...n}$ can be heuristically determined using a Gibbs distribution as discussed in Eq. 7.6. RL based optimization is subjective to that of the reward function and the environment the agent is interacting with. In case of co-adaptation, even though the reward function remains the same, employing Gibbs sampling scheme results in changes to the dynamics of the environment. This very change in the dynamics of the dialogue manager or the user simulation

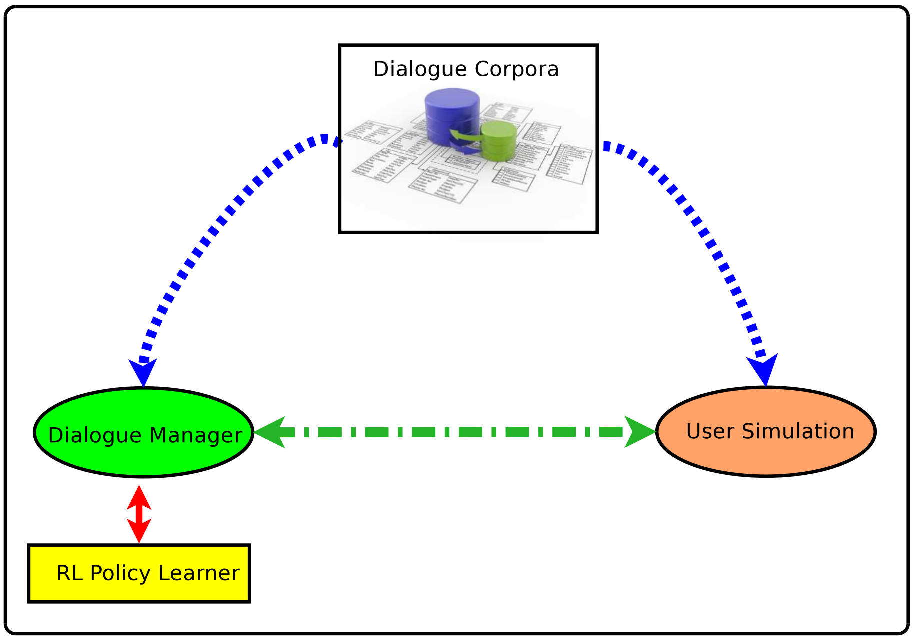


Figure 9.1: *Dialogue optimization using RL and user simulation*

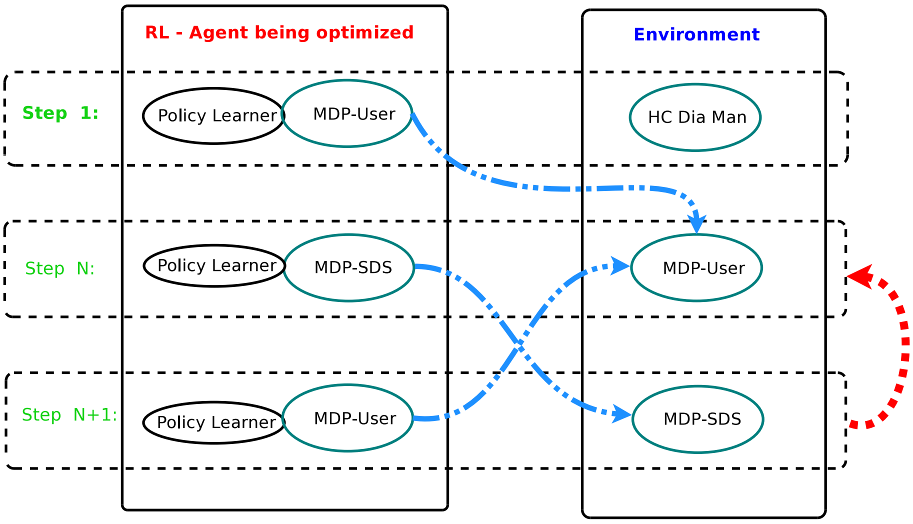


Figure 9.2: *Co-adaptation framework for dialogue evolution*

will yield different user or dialogue policies respectively.

It may be useful to note that, during the process of co-adaptation, the rewards obtained by the agents are back-propagated and thus results in some degree of generalization of dialogue manager as well as user simulation modules. The immediate effect of this would be an opportunity for the dialogue manager and user simulation to cope with unseen situations (which are not observed in the dialogue corpus) provided a good modelization. This in turn will help the dialogue manager to retrieve the real optimal policy from its own capacity. Improvement in dialogue management will result in adaptation of the user simulation. In this case, optimization of User-MDP will yield a different policy than the one observed in the dialogue corpus. This new policy will be adapted to the updated behavior of the dialogue manager and other factors such as ASR errors, *etc.* As a result of co-adaptation, policies which originally may not be present in the corpus[1] can be learned for both the SDS-MDP and the User-MDP. The state of equilibrium, however is defined by the reward function (given suitable reward functions machines will certainly evolve like humans as shown here).

## 9.3 Experimental results and analysis

This section outlines a simple experiment to exhibit the outcome of co-adaptation in spoken dialogue systems. Our primary motivation is to show how co-adaptation can be performed in case of man-machine interfaces and to analyse the possibility of dialogue-evolution. User-MDP policy optimization is performed in two steps: (i) perform IRL on the available dialogue corpus to retrieve the reward function and (ii) use the retrieved reward function obtained from the data to perform co-adaptation. However, dialogue optimization (for SDS-MDP) is performed using a hand-crafted reward function.

### 9.3.1 2-Slot restaurant information SDS

The dialogue problem studied for experimental purposes is the restaurant information system (only with two slots). The dialogue manager has to seek and

[1] Some initial corpus available for user and dialogue manager policy optimization.

obtain user preferences for price-range and location of restaurants in the city. The state of the SDS-MDP (dialogue manager modelled as an MDP) involves 3 dimensions: (i) 0-2 (corresponding to the price-range; 0 - slot is not filled, 1 - slot is filled but yet to be confirmed, 2 - slot is filled and confirmed) (ii) 0-2 (corresponding to the location) and (iii) 0-1 (indicates whether the user has performed negation: indirect measure of channel noise). Dialogue acts include: *ask-slot1, ask-slot2, ask-all-slots, confirm1, confirm2, confirm-both, close-dialogue, as well as 2 implicit confirmation acts.* The state of the User-MDP involves 4 dimensions: (i) 0-9 (represents the action performed by the dialogue manager), (ii) 0-2 (corresponding to the price-range), (iii) 0-2 (corresponding to the location) and (iii) 0-1 (indicates whether speech recognition errors have occurred)[1]. List of user acts includes: *provide-slot1, provide-slot2, provide-all-slots, confirm1, confirm2, confirm-all-slots, negate, remain-silent and hangUp.* The same reward function was used for both SDS-MDP and User-MDP (positive reward of 20 for each correctly filled slot and bonus reward of 60 for successful task completion) and the discount factor is set to 0.95.

### 9.3.2 Co-adaptation results and analysis

To begin with, hand-crafted (HC) policies for both SDS-MDP and User-MDP are defined (these policies attempt to ask/provide and confirm/ascertain one slot after the other) so as to generate data from which a user simulation will be learnt (simulating the acquisition of a corpus). During the experiments dialogue manager and user policy optimization are carried out using Least Squares Policy Iteration (LSPI) [Lagoudakis & Parr, 2003], considering its sample efficiency (recall the relative ease of employing incremental LSPI [Geramifard *et al.*, 2006]). Co-adaptation is performed as explained in Figure 9.2. First user optimization is performed using the SDS-MDP with hand-crafted policy. The resulting user policy was then used to perform dialogue management optimization. During the initial stages of the experiment, the channel noise is set to zero. However, the level of noise is varied after initial experiments. At the end of each step a dia-

[1] This field is set by the dialogue management engine based on a error probability introduced later in this section

logue/user policy is generated. The following are a set of dialogue episodes are generated from these retrieved policies:

```
==============================================
Step 1: HC-DialogueManager vs Train-RL-User
==============================================
  UserState: AskSlot_1 0 0 0
    UserAct: provide_slot_1
  UserState: ExpConfirm_1 1 0 0
    UserAct: confirm_slot_1
  UserState: AskSlot_2 2 0 0
    UserAct: provide_slot_2
  UserState: ExpConfirm_2 2 1 0
    UserAct: confirm_slot_2
  UserState: CloseDia 2 2 0
    UserAct: hangUp
==============================================
Step 2: Train-RL-DialogueManager vs RL-User
==============================================
  DiaState: 0 0 0   DiaAct: AskSlot_1
    UserResponse: provide_slot_1
  DiaState: 1 0 0   DiaAct: ExpConfirm_1
    UserResponse: confirm_slot_1
  DiaState: 2 0 0   DiaAct: AskSlot_2
    UserResponse: provide_slot_2
  DiaState: 2 1 0   DiaAct: ExpConfirm_2
    UserResponse: confirm_slot_2
  DiaState: 2 2 0   DiaAct: CloseDia
    UserResponse: hangUp
==============================================
Step 3: RL-DialogueManager vs Train-RL-User
==============================================
  UserState: AskSlot_1 0 0 0
    UserAct: provide_slot_1
  UserState: ExpConfirm_1 1 0 0
    UserAct: confirm_slot_1
  UserState: AskSlot_2 2 0 0
    UserAct: provide_slot_2
  UserState: ExpConfirm_2 2 1 0
    UserAct: confirm_slot_2
  UserState: CloseDia 2 2 0
    UserAct: hangUp %
===============================================
Step 4: Train-RL-DialogueManager vs RL-User
===============================================
  DiaState: 0 0 0   DiaAct: AskAllSlots
    UserResponse: provide_all_slots
  DiaState: 1 1 0   DiaAct: ExpConfirmAll
    UserResponse: confirm_all_slots
  DiaState: 2 2 0   DiaAct: CloseDia
    UserResponse: hangUp
===============================================
```

*Co-adaptation in dialogue systems (0% ASR error rate)*

The set of dialogues presented here, showcases the possibility of co-adaptation in dialogue systems. Dialogue episodes generated using policies learned from Step 3 and Step 4 clearly indicate that co-adaptation of dialogue management engine and user simulation has indeed happened (observe change in behavior of dialogue manager and user simulation when there is no noise). More interesting aspect of this result is the fact that such optimal policies can be learned even though they were not observed in the dialogue corpus (recall that the hand-crafted policies used in Step 1 only used simple dialogue acts and user acts). This result can be attributed towards the generalization ability of dialogue manager and user simulation when casted as interacting MDPs. It may be useful to note that Gibbs sampling is not introduced in Step 2, since the focus was to learn a basic dialogue policy (similar to the hand-crafted dialogue manager used in Step 1). Even though Gibbs sampling of dialogue policy (from Step 2) was employed in Step 3, the dialogue episode may look similar to that of Step 1. In this case,

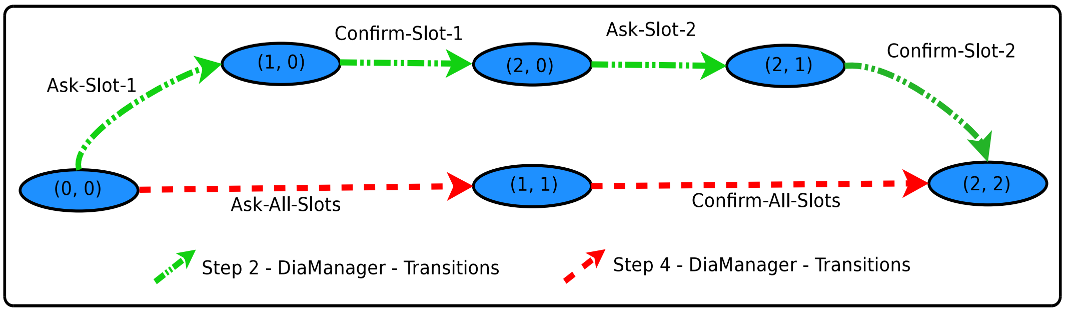


Figure 9.3: *Change in dialogue trajectory due to (error-free) co-adaptation*

the evolution of the user simulation is invisible because the episode presented here is pure greedy interaction between the dialogue policy from Step 2 and user policy from Step 3. However, this invisible evolution of user simulation in Step 3 eventually contributes to the evolution of the dialogue policy in Step 4. Changes in transitions due to co-adaptation of the dialogue management engine is shown in Figure 9.3.

In case of dialogue management, the ability of optimization schemes and resulting policies to cope with ASR channel noise is an important aspect. Therefore, as a next step artificial noise (error model) is introduced, in order to study its effects on the co-adaptation framework. Ideally speaking, if there is some amount of channel noise, performing complex user action (where more information is exchanged) will exponentially increase the possibility for a speech recognition error. Having this in mind, 40% error is introduced when the user simulation performs a complex user action (*i.e.* provide-all-slots). The error field in the user state is set to 1 by the dialogue manager when a confirmation dialogue act is performed. The following are a set of dialogue episodes generated from these retrieved policies:

```
==============================================
Step 1: HC-DialogueManager vs Train-RL-User
==============================================
  UserState: AskSlot_1 0 0 0
    UserAct: provide_slot_1
  UserState: ExpConfirm_1 1 0 0
    UserAct: confirm_slot_1
  UserState: AskSlot_2 2 0 0
    UserAct: provide_slot_2
  UserState: ExpConfirm_2 2 1 0
    UserAct: confirm_slot_2
  UserState: CloseDia 2 2 0
    UserAct: hangUp
==============================================
Step 2: Train-RL-DialogueManager vs RL-User
==============================================
  DiaState: 0 0 0   DiaAct: AskSlot_1
    UserResponse: provide_slot_1
  DiaState: 1 0 0   DiaAct: ExpConfirm_1
    UserResponse: confirm_slot_1
  DiaState: 2 0 0   DiaAct: AskSlot_2
    UserResponse: provide_slot_2
  DiaState: 2 1 0   DiaAct: ExpConfirm_2
    UserResponse: confirm_slot_2
  DiaState: 2 2 0   DiaAct: CloseDia
    UserResponse: hangUp
==============================================
Step 3: RL-DialogueManager vs Train-RL-User
==============================================
  UserState: AskSlot_1 0 0 0
    UserAct: provide_slot_1
  UserState: ExpConfirm_1 1 0 0
    UserAct: confirm_slot_1
  UserState: AskSlot_2 2 0 0
    UserAct: provide_slot_2
  UserState: ExpConfirm_2 2 1 0
    UserAct: confirm_slot_2
  UserState: CloseDia 2 2 0
    UserAct: hangUp %
===============================================
Step 4: Train-RL-DialogueManager vs RL-User
===============================================
  DiaState: 0 0 0   DiaAct: AskSlot_1
    UserResponse: provide_slot_1
  DiaState: 1 0 0   DiaAct: ExpConfirm_1
    UserResponse: confirm_slot_1
  DiaState: 2 0 0   DiaAct: AskSlot_2
    UserResponse: provide_slot_2
  DiaState: 2 1 0   DiaAct: ExpConfirm_2
    UserResponse: confirm_slot_2
  DiaState: 2 2 0   DiaAct: CloseDia
    UserResponse: hangUp
===============================================
```

*Co-adaptation in dialogue systems (40% ASR error rate)*

It can be observed from the above set of dialogues that the dialogue policy and the user policy remain the same even after co-adaptation. It may be useful to note that due to the presence of ASR channel noise, using complex user acts will result in frequent negations, thus both the SDS-MDP and the User-MDP settles with a policy which chooses to perform simple user/dialogue acts. Given the same scenario real users will tend to act in a similar fashion and thus evolve to showcase complex behaviors if the conditions are favourable (as shown in the case of 0% ASR error rate); if not stick to safe means of communication. These experimental results collectively show that the co-adaptation framework proposed here is indeed an effective means for facilitating dialogue evolution.

# Chapter 10

# Conclusion

As mentioned in Chapter 1, the primary focus of this manuscript is twofold: (i) sample efficient dialogue optimization and (ii) IRL-based user simulation. A majority of RL algorithms (recall Section 2.3.2) used to perform dialogue optimization makes inefficient use of training samples. This in turn enforces the necessity for user simulators in order to cope with the data requirement challenge. However, user simulators being computational models introduce some degree of modelling errors. Also as indicated in Chapter 4 the primary focus while user modelling has always been on reproducing the reference dialogue corpus. This results in simulating some non-existent user behavior as outlined in Section 4.3.

Given the current challenges in the dialogue domain, Part II of this manuscript adapts a set of sample efficient algorithms to perform dialogue policy optimization. Also considering the impact of user simulators on the dialogue optimization process, a more realistic user modelling technique based on IRL has been proposed in Part III. The following chapter is organized as follows: To begin with Section 10.1 derives a conclusion on the effectiveness of ADP and KTD algorithms for dialogue optimization. Following which Section 10.2 summarize the effects of user modelling using IRL. Section 10.3 revisits some of the advantages of treating user modelling as a sequential decision making problem and solving it using IRL. Eventually, Section 10.4 outlines the directions of future works.

| Algorithm | Type | ModelReq | Learning mode | UncertaintyInfo |
|---|---|---|---|---|
| Value Iteration | DP | Yes | Model based | Not available |
| Policy Iteration | DP | Yes | Model based | Not available |
| FVI † | ADP | No | Batch method | Not available |
| LSPI | ADP | No | Batch method | Not available |
| Sparse-FVI † | Sparse-ADP | No | Batch method | Not available |
| Sparse-LSPI † | Sparse-ADP | No | Batch method | Not available |
| $Q$-learning | TD-RL | No | Online/Off-policy | Not available |
| SARSA | TD-RL | No | Online/On-policy | Not available |
| KTD-$Q$ † | KTD-RL | No | Online/Off-policy | Available |
| KTD-SARSA † | KTD-RL | No | Online/On-policy | Available |

Table 10.1: *Algorithms for solving SDS-MDP and User-MDP*

## 10.1 Sample efficient policy optimization schemes

In Chapters 5 and 6, a set of ADP, Sparse-ADP and KTD-based algorithms were introduced and adapted to perform dialogue policy optimization. ADP algorithms together can be classified as batch methods (since they learn from a fixed set of data set). Whereas, RL algorithms can be grouped under several classes based on the difference between the control policy and the policy used for data generation (recall the taxonomy of RL algorithms presented in Section 3.4.4). Table 10.1 lists a set of algorithms studied in this manuscript. Note that the † symbol in the table indicates that the corresponding algorithm is adapted to perform dialogue optimization for the first time.

Experimental set-up presented in Section 5.7 studied the effectiveness of ADP (fitted-$Q$ and LSPI) and Sparse-ADP (sparse-fitted-$Q$ and sparse-LSPI) when applied to perform dialogue optimization. Based on the experimental results we can now conclude that these methods are indeed sample efficient (in comparison to conventional RL algorithms). However, it is important to note that these methods need some training corpus to learn in a batch setting. They merely focus on learning an optimal policy from the corpus and provide no scope for improving this initial policy once the learning is terminated. However, this offers an opportunity to learn an initial policy using these sample efficient algorithms and further improve the policy in a online setting. Even though they promise to be a good alternative, ADP-based algorithms provide no guarantees to retrieve an (near) optimal policy.

Sample efficiency of KTD-based algorithms (KTD-$Q$ and KTD-SARSA) are studied using a set of experiments outlined in Section 6.5. Experimental results obtained in the dialogue domain indicate that these algorithms make efficient use of training samples. Since the KTD framework maintains uncertainty information with regard to $Q$-value estimates, smart exploration schemes (which accelerates learning) can be employed. Since they fall under the class of online RL algorithms they can be used to learn in an off-policy or on-policy setting in a controlled or uncontrolled manner. An initial policy learned from the available dialogue corpus (either using ADP-based batch methods or using online/off-policy/uncontrolled manner) and can be further improved in an online/on-policy/controlled setting. It is important to note that during online/on-policy learning in case of the dialogue domain policy changes can result in adverse system behaviors and thus may cause incontinence to real users (leaving users unsatisfied). Thus it is always better to learn a fairly good initial in a off-policy setting and then switch to on-policy setting for improving the policy further.

## 10.2 User simulation using IRL

IRL-based user simulation in dialogue systems was introduced in Chapter 7. This novel method for user modelling can be seen as an effort towards simulating dialogue trajectories rather than dialogue transitions. To begin the task of user simulation is treated as a sequential decision making problem and casted as an MDP. In order to optimize the User-MDP using RL methods some reward function is required. Learning this reward function from the dialogue corpus using IRL provides an implicit mechanism for simulating real user behavior (observed in the corpus). Experimental results presented in Section 7.3 showcase that the estimated policies for User-MDP indeed simulate a behavior similar to the reference dialogue corpus. Even though the policies retrieved are deterministic, stochastic user behavior can be derived by employing a policy mixer based on Gibbs sampling.

Discounted feature vectors (see Eq. 8.1) provide new possibilities for analysing the user behavior. For instance, since it provides a compact summary of dialogue trajectories (irrespective of their length), it can be used for clustering user behaviors (recall Section 8.1.2). Experimental results presented in Section 8.2 showcase that behavior clustering can be used to identify and differentiate user behaviors. This can be seen as an important result in the dialogue domain for two reasons: (i) being an automatic scheme, behavior clustering scheme can be employed as a preprocessing step on the reference dialogue corpus and (ii) once different user behaviors are identified, behaviors corresponding to every user group can be simulated. This overcomes the critical shortcomings of existing methods for user modelling which focus on simulating some generic non-existent user behavior (in the process of reproducing the reference dialogue corpus). Identifying and/or simulating different user behaviors is a crucial task in order to adapt a dialogue policy to all (observable) user groups rather than adapting it to a generic user behavior.

## 10.3 Co-adaptation in spoken dialogue systems

Treating the task of user simulation as a sequential decision making problem provides some novel opportunities for generalization of user behavior and co-adaptation in dialogue systems. Mutual adaptation (at the level of words, amount of information exchanged) can be frequently observed in human-human communication. Modelling user as an MDP and making it interact with a dialogue manager modelled as an MDP provides a framework for co-adaptation in dialogue systems. Experimental results presented in Section 9.3 outlines the effects of co-adaptation in the presence and absence of ASR channel noise. These results indicate that it is possible to learn an initial policy (for both SDS-MDP and User-MDP) from the reference dialogue corpus and then learn further policies as a result of co-adaptation between the dialogue manager and user simulator. In case of the dialogue domain co-adaptation can be advantageous in several ways: (i) by providing a natural means for evolution, co-adaptation shifts human/machine-machine interaction closer towards human-human interaction, (ii) possibility to learn complex policies which are originally not observed in the reference dialogue corpus, (iii) provides a framework for introducing evolution at the word-level, intention-level *etc.*

## 10.4 Future works

The works presented in this manuscript primarily aims at estimating user adaptive policies for spoken dialogue management. Such policies are necessary in order to guarantee user satisfaction in human-machine interaction. With regard to the future works, there are several possibilities relating to both sample efficient dialogue optimization and/or IRL user simulation. The following section presents some directions for future works.

### 10.4.1 Model selection in reinforcement learning

Sample efficiency of ADP and KTD-based algorithms when adapted to dialogue optimization has been showcased with the help of experimental results. In the first place these algorithms were introduced to suppress the necessity for user simulators. However, user simulators were still employed for evaluation purposes. The primary reason for their continued use is the ease with which policy evaluation can be performed by using simulators. They provide an automatic and fast scheme for evaluating the quality of a dialogue policy. Thus, one interesting direction of future work would be to explore the possibility for policy evaluation using the available dialogue corpus. In order to envision this, model selection schemes [Farahmand & Szepesvári, 2011] can be used to quantify the quality of the estimated (dialogue or user) policies.

### 10.4.2 IRL-based dialogue optimization

So-far in this manuscript IRL was used for the purpose of user simulation. This can as well be extended for spoken dialogue management. Although, such an extension would need the data to reflect the optimal behavior so that it can be reproduced directly. For instance, if the dialogue corpus consists of human-human (or Wiz-of-Oz) dialogues, an optimal dialogue management policy can be reproduced by means of imitation. However, this option turns out to be challenging if the dialogue corpus does not reflect the optimal behavior (for instance if the dialogues are generated using human-machine setting where the machine employs some basic hand-crafted policy). In such cases it is necessary to employ IRL methods which focus on retrieving the task related information at a generic level. In other words it is important predict the structure of the task rather than focusing on how the task is accomplished. Also one other direction of future work is to scale up the IRL-based methods for complex dialogue management problems.

### 10.4.3 Transfer learning in dialogue systems

Transfer learning is the field of study which explores the possibility of using knowledge gained in solving one task to solve another but relevant task. In recent years there has been a significant interest to extend transfer learning for RL settings [Taylor & Stone, 2009]. In general the dialogue domain presents an ideal case for employing transfer learning. For instance (dialogue management or user simulation) skills learned for handling a 3 slots restaurant information SDS can be used to handle a 5 slots flight ticket booking SDS. One possible way to perform this task is to first learn the structural relation between the state-action space and the reward function for the 3 slots problem space. Once this relation is learned it can then be transformed to the 5 slots dialogue problem space.

### 10.4.4 Scaling up IRL-based methods for SDS

Experimental analysis used for validating the proposed IRL-based user simulation focused on a relatively simple slot filling dialogue problem. Even though the results showcased the possibility for user simulation, the true potential of IRL-based methods is yet to be revealed. In the first place IRL-based methods were introduced to cope with complex decision making problems where it is impractical to manually specify a reward function. Thus the IRL-framework in itself is capable and in fact is meant for dealing with complex dialogue problems. However, it was necessary to explore the possibility of using IRL in the dialogue domain. Our initial results are promising and have opened several new arenas for research. Thus an obvious direction of future work would be to scale up IRL methods for complex dialogue management problems and corresponding user simulations. One of the critical factors during scaling up would be to identify a suitable set of state features for SDS-MDP and User-MDP. However, there exist a set of automatic schemes (such as [Lefèvre & de Mori, 2007]) which can learn a suitable state representation. Such scheme can be used to cast the dialogue problem in-terms of SDS-MDP and User-MDP.

### 10.4.5 Generalization of user behaviors

Treating user simulation as a sequential decision making problem offered an opportunity to cast the task as an MDP. This in turn opened up a possibility for generalization of user simulations. RL based optimization of User-MDPs back-propagate the rewards and thus help the user model to generalize the behavior for scenarios which are originally not seen in the dialogue corpus. In fact the results presented while outlining co-adaptation in dialogue systems is an exact consequence of behavior generalization. However, to what extent user behaviors can be generalized and how the quality of such generalizations can be quantified is a open question to be answered in the near future.

### 10.4.6 Dialogue adaptation to behavior specific users

One of the primary advantages of using IRL-based methods for user modelling is their ability to learn multiple behavior specific user simulation rather than learning one generic behavior. This possibility was showcased by means of experimental result in Section 8.2. Now given the availability of multiple simulators, a obvious question to be addressed is: how to employ them for dialogue optimization. One possibility would be to randomly switch the user simulators during the optimization process and have one dialogue policy which is expected to be adaptive to all the user behaviors (as shown in [Chandramohan & Pietquin, 2010]). What needs to be determined is in what way the policy learned by switching the user simulators will be different from the one learned using other existing user simulation methods (which exhibit a generic behavior)? The other possibility would be to learn one dialogue policy for every user behavior and have a mechanism which decides which policy to use during live interaction with users. However, what mechanism should be employed for policy selection is an open question. One other direction of future work would be to study the possibility for predicting missing user behaviors (using other available behaviors) and derive schemes to automatically generate data reflecting user behaviors which are not available in the dialogue corpus.

# References

Abbeel, P. & Ng, A. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proc. of ICML*, Banff, Alberta (Canada). 109

Ai, H. & Litman, D. (2008). Assessing dialog system user simulation evaluation measures using human judges. In *Proc. of the 46th meeting of the Association for Computational Linguistics*, 622–629, Columbus (OH). 101

Allen, J. (1995). *Natural language understanding (2nd ed.)*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA. 12

Altman, I. & Taylor, D. (1973). *Social penetration: The development of interpersonal relationships*. Holt, Rinehart and Winston. 134

Astrom, K.J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications, vol. 10, pp. 174–205*. 28

Austin, J.L. (1975). *How to Do Things with Words: Second Edition (William James Lectures)*. Harvard University Press, 2nd edn. 12

Baxter, L. & Braithwaite, D. (2008). *Engaging Theories in Interpersonal Communication: Multiple Perspectives*. Sage Publications. 134

Bellman, R. (1957a). *Dynamic Programming*. Dover Publications, sixth edn. 5, 27, 28, 30, 32

Bellman, R. (1957b). A markovian decision process. *Journal of Mathematics and Mechanics, vol. 6, pp. 679–684*. 23

Bellman, R. & Dreyfus, S. (1959). Functional approximation and dynamic programming. *Mathematical Tables and Other Aids to Computation*, **13**, 247–251. 2, 62, 64

Berger, C.R. (1986). Uncertain outcome values in predicted relationships: Uncertainty reduction theory then and now. In H.C. Research, ed., *Human Communication Research*, vol. 13, 34–38. 134

Boularias, A., Kober, J. & Peters, J. (2011). Relative entropy inverse reinforcement learning. *Journal of Machine Learning Research - Proceedings Track*, **15**, 182–189. 109, 135

Bradtke, S.J. & Barto, A.G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, **22**, 33–57. 67

Chandramohan, S. & Pietquin, O. (2010). User and Noise Adaptive Dialogue Management Using Hybrid System Actions. In *Spoken Dialogue Systems for Ambient Environments*, vol. 6392 of *Lecture Notes in Artificial Intelligence (LNAI)*, 13–24, proc. of IWSDS 2010, Gotemba, Shizuoka (Japan). 151

Chandramohan, S., Geist, M. & Pietquin, O. (2010a). Optimizing Spoken Dialogue Management with Fitted Value Iteration. In *Proc. of InterSpeech 2010*, Makuhari (Japan). 77, 82

Chandramohan, S., Geist, M. & Pietquin, O. (2010b). Sparse Approximate Dynamic Programming for Dialog Management. In *Proc. of SIGDial*, Tokyo (Japan). 82

Chandramohan, S., Geist, M., Lefèvre, F. & Pietquin, O. (2011a). User Simulation in Dialogue Systems using Inverse Reinforcement Learning. In *Proc. of Interspeech 2011*, Florence (Italy). 2, 122

Chandramohan, S., Geist, M. & Pietquin, O. (2011b). Apprentissage par Renforcement Inverse pour la Simulation d'Utilisateurs dans les Systèmes de Dialogue. In *Sixièmes Journées Francophones de Planification, Décision et Apprentissage pour la conduite de systèmes (JFPDA 2011)*, 7 pages, Rouen (France).

Chandramohan, S., Geist, M., Lefèvre, F. & Pietquin, O. (2012a). Behavior Specific User Simulation in Spoken Dialogue Systems. In *Proc. of the IEEE ITG Conference on Speech Communication (to appear)*, Braunschweig, Germany.

Chandramohan, S., Geist, M., Lefèvre, F. & Pietquin, O. (2012b). Clustering Behaviors of Spoken Dialogue Systems Users. In *Proc. of the 37th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2012)*, Kyoto (Japan).

Chandramohan, S., Geist, M., Lefèvre, F. & Pietquin, O. (2012c). Regroupement non-supervisé d'utilisateurs par leur comportement pour les systèmes de dialogue parlé. In *Journées Francophones de Planification, Décision et Apprentissage pour la conduite de systèmes (JFPDA 2012)*, Nancy (France).

Chung, G., Seneff, S. & Wang, C. (2003). Automatic acquisition of names using speak and spell mode in spoken dialogue systems. In *Proc. of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, 32–39, Association for Computational Linguistics, Stroudsburg, PA, USA. 134

Cuayáhuitl, H., Renals, S., Lemon, O. & Shimodaira, H. (2005). Human-computer dialogue simulation using hidden markov models. In *Proc. of the Automatic Speech Recognition Workshop (ASRU)*, Cancun (Mexico). 53

Daubigney, L., Gasic, M., Chandramohan, S., Geist, M., Pietquin, O. & Young, S. (2011). Uncertainty management for on-line optimisation of a POMDP-based large-scale spoken dialogue system. In *Proc. of Interspeech 2011*, 1301–1304, Florence (Italy). 133

Dearden, R., Friedman, N. & Russell, S.J. (1998). Bayesian Q-Learning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI)*, 761–768. 91

DODDINGTON, G. (2002). Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proc. of the Human Language Technology Conference (HLT)*, San Diego (CA). 54

DUTOIT, T. (1997). *An introduction to text-to-speech synthesis*. Kluwer Academic Publishers, Norwell, MA, USA. 13

ECKERT, W., LEVIN, E. & PIERACCINI, R. (1997a). User modeling for spoken dialogue system evaluation. In *Proc. of the Automatic Speech Recognition Workshop (ASRU)*, Santa Barbara (CA). 47, 48, 101

ECKERT, W., LEVIN, E. & PIERACCINI, R. (1997b). User Modeling for Spoken Dialogue System Evaluation. In *Proc. of ASRU*, 80–87. 132

ENGEL, Y., MANNOR, S. & MEIR, R. (2004). The Kernel Recursive Least Squares Algorithm. *IEEE Transactions on Signal Processing, vol. 52(8), pp. 2275–2285*. 64, 68, 69, 70

FARAHMAND, A.M. & SZEPESVÁRI, C. (2011). Model selection in reinforcement learning. *Machine Learning*, 1–34. 74, 149

FERGUSON, G., ALLEN, J. & MILLER, B. (1996). Trains-95: Towards a mixed-initiative planning assistant. In *in Proceedings of the 3rd Conference on AI Planning Systems*. 20

FRAMPTON, M. & LEMON, O. (2009). Recent research advances in reinforcement learning in spoken dialogue systems. *Knowledge Eng. Review*, **24**, 375–408. 13

GASIC, M., JURCICEK, F., THOMSON, B., YU, K. & YOUNG, S. (2011). Online policy optimisation of spoken dialogue systems via live interaction with human subjects. In *Proc. of ASRU 2011*, Hawaii (USA). 133

GEIST, M. (2009). *Optimisation des chanes de production dans l'industrie sidérurgique : une approche statistique de l'apprentissage par renforcement*. Phd thesis in mathematics, Université Paul Verlaine de Metz (en collaboration avec Supélec, ArcelorMittal et l'INRIA). 92

GEIST, M. & PIETQUIN, O. (2010a). A Brief Survey of Parametric Value Function Approximation. Tech. rep., Supélec - Metz (France). 62

GEIST, M. & PIETQUIN, O. (2010b). Kalman Temporal Differences. *Journal of Artificial Intelligence Research (JAIR)*, **39**, 483–532. 2, 62, 85

GEIST, M. & PIETQUIN, O. (2010c). Managing Uncertainty within Value Function Approximation in Reinforcement Learning. In *Active Learning and Experimental Design workshop (collocated with AISTATS 2010)*, Sardinia, Italy. 92

GEIST, M., PIETQUIN, O. & FRICOUT, G. (2008). Kalman Temporal Differences: Uncertainty and Value Function Approximation. In *NIPS Workshop on Model Uncertainty and Risk in Reinforcement Learning*, Vancouver (Canada). 91

GEIST, M., PIETQUIN, O. & FRICOUT, G. (2009). Kalman Temporal Differences: the deterministic case . In *IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2009)*, 185–192, Nashville (TN, USA). 87

GEORGILA, K., HENDERSON, J. & LEMON, O. (2005). Learning user simulations for information state update dialogue systems. In *Proc. Interspeech '05*, Lisbon (Portugal). 47, 48

GEORGILA, K., HENDERSON, J. & LEMON, O. (2006). User simulation for spoken dialogue systems: Learning and evaluation. In *Proc. International Conference on Spoken Language Processing (Interspeech/ICSLP)*, Pittsburgh (PA). 50, 74

GERAMIFARD, A., BOWLING, M. & SUTTON, R.S. (2006). Incremental least-squares temporal difference learning. In *Proc. of AAAI*, 356–361, AAAI Press. 114, 138

GERGONNE, J. (1974). The application of the method of least squares to the interpolation of sequences. *Historia Mathematica*, **1**, 439 – 447. 62

GLASS, J. & SENEFF, S. (2003). Flexible and personalizable mixed-initiative dialogue systems. In *Proc. of the HLT-NAACL 2003 workshop on Research directions in dialogue processing - Volume 7*, 19–21, Association for Computational Linguistics, Stroudsburg, PA, USA. 134

GÖTZE, J., SCHEFFLER, T., ROLLER, R. & REITHINGER, N. (2010). User simulation for the evaluation of bus information systems. In *Proc. IEEE Spoken Language Technology Workshop (SLT)*, Berkeley (CA). 46

HAHN, S., DINARELLI, M., RAYMOND, C., LEFÈVRE, F., LEHNEN, P., DE MORI, R., MOSCHITTI, A., NEY, H. & RICCARDI, G. (2011). Comparing stochastic approaches to spoken language understanding in multiple languages. *Audio, Speech, and Language Processing, IEEE Transactions on*, **19**, 1569 –1583. 22

HENDERSON, J. & LEMON, O. (2008). Mixture model pomdps for efficient handling of uncertainty in dialogue management. In *Proc. of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, HLT-Short '08, 73–76, Association for Computational Linguistics, Stroudsburg, PA, USA. 22

JANARTHANAM, S. & LEMON, O. (2009). Learning Adaptive Referring Expression Generation Policies for Spoken Dialogue Systems using Reinforcement Learning. In *Proceedings SemDial'09, Stockholm*. 25

JELINEK, F. (1998). *Statistical Methods for Speech Recognition*. The MIT Press. 22

JUNG, S., LEE, C., KIM, K., JEONG, M. & LEE, G.G. (2009). Data-driven user simulation for automated evaluation of spoken dialogue systems. *Computer Speech and Language*, **23**, 479–509. 46, 54

JNSSON, A. (1993). Dialogue management for natural language interfaces. Tech. rep., THE UNIVERSITY OF QUEENSLAND. 19

KAELBLING, L.P., LITTMAN, M.L. & MOORE, A.W. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, **4**, 237–285. 23

KALMAN, R.E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, **82**, 35–45. 87, 88

KEIZER, S., GASIC, M., JURCICEK, F., MAIRESSE, F., THOMSON, B., YU, K. & YOUNG, S. (2010). Parameter estimation for agenda-based user simulation. In *Proc. of the SIGDIAL 2010 Conference*, 116–123. 124, 125

KEIZER, S., ROSSIGNOL, S., CHANDRAMOHAN, S. & PIETQUIN, O. (2012). User Simulation in the Development of Statistical Spoken Dialogue Systems. In *Data driven methods for Adaptive Spoken Dialogue Systems*, Springer-Verlag New York Inc (To appear), (To appear).

KOLTER, J.Z. & NG, A.Y. (2009). Near-Bayesian Exploration in Polynomial Time. In *Proceedings of the 26th international conference on Machine learning (ICML 09)*, ACM, New York, NY, USA. 91, 92

KULLBACK, S. & LEIBLER, R. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, **22**, 79–86. 53, 127

LAGOUDAKIS, M.G. & PARR, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, **4**, 1107–1149. 66, 114, 138

LARSSON, S. & TRAUM, D. (2000). Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering, vol. 6, pp 323–340*. 16

LEE, A. & PRZYBOCKI, M. (2005). NIST Machine translation evaluation official results. Official release of automatic evaluation scores for all submissions. 54

LEFÈVRE, F. (2007). Dynamic bayesian networks and discriminative classifiers for multi-stage semantic interpretation. Hawaii, USA. 22

LEFÈVRE, F. & DE MORI, R. (2007). Unsupervised state clustering for stochastic dialog management. In *Proc. of ASRU*, Kyoto (Japan). 151

LEMON, O. (2011). Learning what to say and how to say it: Joint optimisation of spoken dialogue management and natural language generation. *Computer Speech & Language*, **25**, 210 – 221. 25

LEMON, O. & PIETQUIN, O. (2007). Machine learning for spoken dialogue systems. In *Proc. of InterSpeech'07*, Belgium. 1

LEMON, O., GEORGILA, K., HENDERSON, J. & STUTTLE, M. (2006). An ISU dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the TALK in-car system. In *Proc. of EACL'06*, Morristown, NJ, USA. 1, 19, 74, 76

LEVIN, E. & PIERACCINI, R. (1998). Using markov decision process for learning dialogue strategies. In *Proc. ICASSP'98, Seattle (USA)*. 13, 23, 42

LEVIN, E., PIERACCINI, R. & ECKERT, W. (2000). A Stochastic Model of Human-Machine Interaction for learning dialog Strategies. *IEEE Transactions on Speech and Audio Processing, vol. 8, pp. 11–23*. 2, 24

LI, L., BALAKRISHNAN, S. & WILLIAMS, J. (2009). Reinforcement Learning for Dialog Management using Least-Squares Policy Iteration and Fast Feature Selection. In *Proc. of the International Conference on Speech Communication and Technologies (InterSpeech'09)*, Brighton (UK). 63, 68, 77, 82

LÓPEZ-CÓZAR, R., CALLEJAS, Z. & MCTEAR, M.F. (2006). Testing the performance of spoken dialogue systems by means of an artificially simulated user. *Artificial Intelligence Review*, **26**, 291–323. 101

MEHTA, N., GUPTA, R., RAUX, A., RAMACHANDRAN, D. & KRAWCZYK, S. (2010). Probabilistic Ontology Trees for Belief Tracking in Dialog Systems. In *Proc. of the SIGDIAL 2010 Conference*, 37–46, Association for Computational Linguistics, Tokyo, Japan. 103

NG, A.Y. & RUSSELL, S. (2000). Algorithms for inverse reinforcement learning. In *Proc. of ICML*, Stanford (CA). 2, 107

PAPINENI, K., ROUKOS, S., WARD, T. & ZHU, W. (2002). BLEU: A method for automatic evaluation of machine translation. In *Proc. of the 40th Annual Meeting on Association for Computational Linguistics (ACL)*, Philadelphia (PA). 54

PARK, J. & SANDBERG, I. (1991). Universal approximation using radial-basis-function networks. *Neural computation*, **3**, 246–257. 62, 68

PIETQUIN, O. (2005). A probabilistic description of man-machine spoken communication. In *Proc. of ICME 2005*, 410–413, Amsterdam (The Netherlands). 102

PIETQUIN, O. (2006). Consistent goal-directed user model for realistic man-machine task-oriented spoken dialogue simulation. In *Proc. of ICME*, 425–428, Toronto (Canada). 24

PIETQUIN, O. & DUTOIT, T. (2006). A probabilistic framework for dialog simulation and optimal strategy learning. *IEEE Transactions on Audio, Speech & Language Processing, 14(2): 589-599*. 47

PIETQUIN, O. & HASTIE, H. (2011). A survey on metrics for the evaluation of user simulations. *The Knowledge Engineering Review*. 51, 52, 116

PIETQUIN, O., GEIST, M. & CHANDRAMOHAN, S. (2011a). Sample Efficient On-line Learning of Optimal Dialogue Policies with Kalman Temporal Differences. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, Barcelona (Spain). 2

PIETQUIN, O., GEIST, M., CHANDRAMOHAN, S. & FREZZA-BUET, H. (2011b). Sample-Efficient Batch Reinforcement Learning for Dialogue Management Optimization. *ACM Transactions on Speech and Language Processing*, **7**, 7:1–7:21. 2, 63, 74, 78, 82

PIETQUIN, O., ROSSIGNOL, S. & IANOTTO, M. (2009). Training Bayesian networks for realistic man-machine spoken dialogue simulation. In *Proc. of IWSDS 2009*, Irsee (Germany). 49

Pinault, F. & Lefèvre, F. (2011). Semantic graph clustering for pomdp-based spoken dialog systems. In *Proc. of Interspeech*, 1321–1324. 23

Puterman, M.L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience. 5, 27, 31, 34

Rabiner, L. & Juang, B.H. (1993). *Fundamentals of Speech Recognition*. Prentice Hall Signal Processing Series, PTR Prentice-Hall. 12

Reiter, E. & Dale, R. (2000). *Building natural language generation systems*. Cambridge University Press, New York, NY, USA. 13

Rieser, V. (2008). *Bootstrapping Reinforcement Learning-based Dialogue Strategies from Wizard-of-Oz data*. Ph.D. thesis, Saarland University, Department of Computational Linguistics. 24, 54, 102

Rieser, V. & Lemon, O. (2006). Simulations for learning dialogue strategies. In *Proc. of Interspeech 2006*, Pittsburg (PA). 54

Rieser, V. & Lemon, O. (2011). *Reinforcement Learning for Adaptive Dialogue Systems: A Data-driven Methodology for Dialogue Management and Natural Language Generation*. Theory and Applications of Natural Language Processing, Springer-Verlag New York Inc. 1, 132

Roy, N., Pineau, J. & Thrun, S. (2000). Spoken dialogue management using probabilistic reasoning. In *Proc. of the annual meetinf of the Association for Computational Linguistics (ACL'00)*, Morristown, NJ, USA. 13

Sakaguchi, Y. & Takano, M. (2004). Reliability of internal prediction/estimation and its application: I. adaptive action selection reflecting reliability of value function. *Neural Network*, **17**, 935–952. 91

Samuel, A.L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 210–229. 64

Schatzmann, J., Stuttle, M.N., Weilhammer, K. & Young, S. (2005). Effects of the user model on simulation-based learning of dialogue strategies. In *Proc. of ASRU'05, Puerto Rico*. 25, 51, 102, 116

Schatzmann, J., Weilhammer, K., Stuttle, M. & Young, S. (2006a). A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowledge Engineering Review, vol. 21(2), pp. 97–126*. 2, 24

Schatzmann, J., Weilhammer, K., Stuttle, M. & Young, S. (2006b). A survey of statistical user simulation techniques for reinforcement learning of dialogue management strategies. *The Knowledge Engineering Review*, **21**, 97–126. 25, 46, 52

Schatzmann, J., Thomson, B., Weilhammer, K., Ye, H. & Young., S. (2007a). Agenda-based User Simulation for Bootstrapping a POMDP Dialogue System. In *Proc. of HLT NAACL*. 47, 102

Schatzmann, J., Thomson, B., Weilhammer, K., Ye, H. & Young, S. (2007b). Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Proc. of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, Rochester (NY). 50

Seneff, S., Hurley, E., Lau, R., Pao, C., Schmid, P. & Zue, V. (1998). Galaxy-ii: A reference architecture for conversational system development. In *in Proc. ICSLP*, 931–934. 19

Singh, S., Kearns, M., Litman, D. & Walker, M. (1999). Reinforcement learning for spoken dialogue systems. In *Proc. of NIPS, Denver, USA*, Springer. 42

Sondik, E.J. (1978). The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, **26**, 282–304. 23

Strehl, A.L. & Littman, M.L. (2006). An Analysis of Model-Based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*. 91

SUTTON, R. & BARTO, A. (1998). *Reinforcement Learning: An Introduction*. The MIT Press, 3rd edn. 1, 23, 28, 35, 36, 37, 106

TAYLOR, M.E. & STONE, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, **10**, 1633–1685. 150

THEUNE, M. (2003). Natural language generation for dialogue: system survey. Tech. Rep. 2003-2, University of Twente (Netherlands). 25

VAN RIJSBERGEN, C.J. (1979). *Information Retrieval*. Butterworths, London (UK), 2nd edn. 52

VANLEHN, K., JORDAN, P. & LITMAN, D. (2007). D.: Developing pedagogically effective tutorial dialogue tactics: Experiments and a testbed. In *Proc. of SLaTE Workshop on Speech and Language Technology in Education.*. 19

WALKER, M.A., LITMAN, D.J., KAMM, C.A. & ABELLA, A. (1997). PARADISE: A framework for evaluating spoken dialogue agents. In *Proc. of the 35th Annual Meeting of the Association for Computational Linguistics (ACL'97)*, 271–280, Madrid (Spain). 42

WATKINS, C.J.C.H. & DAYAN, P. (1992). Q-learning. *Machine Learning*, **8**, 272–292. 38

WILKS, Y. (2004). Artificial companions. In *Proceedings of the 1st International Workshop on Machine Learning for Multimodal Interaction*, Switzerland. 19

WILLIAMS, J.D. & YOUNG, S. (2007). Partially observable markov decision processes for spoken dialog systems. *Computer Speech and Language, vol. 21(2), pp. 393–422.*. 1, 19, 22, 23

XU, X., HU, D. & LU, X. (2007). Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, **18**, 973–992. 68, 71

ZUKERMAN, I. & ALBRECHT, D. (2001). Predictive statistical models for user modeling. *User Modeling and User-Adapted Interaction*, **11**, 5–18. 52