Learning Dialogue POMDP Model Components from Expert Dialogues

Thèse présentée
à la Faculté des études supérieures et postdoctorales de l’Université Laval
dans le cadre du programme de doctorat en informatique
pour l’obtention du grade de Philosophiæ Doctor (Ph.D.)

DÉPARTEMENT D’INFORMATIQUE ET DE GÉNIE LOGICIEL
FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

2013

©Hamidreza Chinaei, 2013
Résumé

Un système de dialogue conversationnel doit aider les utilisateurs humains à atteindre leurs objectifs à travers des dialogues naturels et efficaces. C’est une tâche toutefois difficile car les langages naturels sont ambiguës et incertains, de plus le système de reconnaissance vocale (ASR) est bruité. À cela s’ajoute le fait que l’utilisateur humain peut changer son intention lors de l’interaction avec la machine. Dans ce contexte, l’application des processus décisionnels de Markov partiellement observables (POMDPs) au système de dialogue conversationnel nous a permis d’avoir un cadre formel pour représenter explicitement les incertitudes, et automatiser la politique d’optimisation. L’estimation des composantes du modèle d’un POMDP-dialogue constitue donc un défi important, car une telle estimation a un impact direct sur la politique d’optimisation du POMDP-dialogue.

Abstract

Spoken dialogue systems should realize the user intentions and maintain a natural and efficient dialogue with users. This is however a difficult task as spoken language is naturally ambiguous and uncertain, and further the automatic speech recognition (ASR) output is noisy. In addition, the human user may change his intention during the interaction with the machine. To tackle this difficult task, the partially observable Markov decision process (POMDP) framework has been applied in dialogue systems as a formal framework to represent uncertainty explicitly while supporting automated policy solving. In this context, estimating the dialogue POMDP model components is a significant challenge as they have a direct impact on the optimized dialogue POMDP policy.

This thesis proposes methods for learning dialogue POMDP model components using noisy and unannotated dialogues. Specifically, we introduce techniques to learn the set of possible user intentions from dialogues, use them as the dialogue POMDP states, and learn a maximum likelihood POMDP transition model from data. Since it is crucial to reduce the observation state size, we then propose two observation models: the keyword model and the intention model. Using these two models, the number of observations is reduced significantly while the POMDP performance remains high particularly in the intention POMDP. In addition to these model components, POMDPs also require a reward function. So, we propose new algorithms for learning the POMDP reward model from dialogues based on inverse reinforcement learning (IRL). In particular, we propose the POMDP-IRL-BT algorithm (BT for belief transition) that works on the belief states available in the dialogues. This algorithm learns the reward model by estimating a belief transition model, similar to MDP (Markov decision process) transition models. Ultimately, we apply the proposed methods on a healthcare domain and learn a dialogue POMDP essentially from real unannotated and noisy dialogues.
Acknowledgement

I am deeply grateful to my thesis advisor, Brahim Chaib-draa, for accepting me as a PhD student in his group, DAMAS, for his “various” supports during my PhD studies, and for being a great advisor. I am also very thankful to my co-advisor, Luc Lamontagne, for his support and helpful discussions in different stages of my PhD studies.

I am highly thankful to Jason D. Williams (now at Microsoft research), with whom I had the opportunity of having an internship at AT&T research. Indeed, Jason’s thesis, papers, and advice, during and after my internship, cleared lots of the obstacles that I faced along my PhD path. I am also grateful to Suhrid Balakrishnan who I met at AT&T research. The regular weekly discussions with Jason and Suhrid at AT&T research were great, fun, and indeed helpful. In addition, I am thankful to my thesis committee members, Joelle Pineau, Olivier Pietquin, Philippe Giguere, and Nadir Belkhiter, for their helpful comments and feedback. I appreciate Joelle for providing me with the SmartWheeler data set on which I performed part of my experiments.

Many thanks to my teammates in DAMAS, with whom I spent most of my PhD life. In addition, I would like to thank Québec and FQRNT (Fonds Québécois de la recherche sur la nature et les technologies) from which I received partial financial support for my PhD studies. In fact, my life in Québec was exciting, fun, and memorable, and I met many many great friends in Québec city. I am particularly thankful to Mahshid, Mehdi, Maxime, Davood, Abdeslam, Jilles, Sara, Patrick, Parvin, and Ethan.

Last but not least, I would like to give my deep gratitude to my family for their never ending support during my life and for the sweet and challenging times that we spent together. To those that were always there to motivate, support, and give their care, I thank you all.
To my mother and in memory of my father
# Contents

Résumé ii

Abstract iii

Acknowledgement iv

Table of Contents vi

List of Figures vii

List of Tables viii

List of Algorithms ix

Notations and acronyms x

1 Introduction 1
   1.1 Approach ................................................. 5
   1.2 Main contributions ..................................... 6
   1.3 Thesis structure ....................................... 8

2 Topic modeling 9
   2.1 Dirichlet distribution .................................. 9
      2.1.1 Exponential distributions ......................... 9
      2.1.2 Multinomial distribution ......................... 10
      2.1.3 Dirichlet distribution ......................... 10
      2.1.4 Example on the Dirichlet distribution ............ 12
   2.2 Latent Dirichlet allocation ........................... 14
   2.3 Hidden Markov models ................................. 18

3 Sequential decision making in spoken dialogue management 23
   3.1 Sequential decision making ........................... 23
      3.1.1 Markov decision processes (MDPs) ................. 25
      3.1.2 Partially observable Markov decision processes (POMDPs) ... 26
3.1.3 Reinforcement learning ................................................. 29
3.1.4 Solving MDPs/POMDPs ................................................. 30
  3.1.4.1 Policy iteration for MDPs .................................. 30
  3.1.4.2 Value iteration for MDPs .................................. 31
  3.1.4.3 Value iteration for POMDPs ................................. 32
  3.1.4.4 Point-based value iteration for POMDPs ................. 35

3.2 Spoken dialogue management ......................................... 38
  3.2.1 MDP-based dialogue policy learning ......................... 39
  3.2.2 POMDP-based dialogue policy learning ....................... 40
  3.2.3 User modeling in dialogue POMDPs .......................... 43

4 Dialogue POMDP model learning ....................................... 46
  4.1 Introduction .......................................................... 46
  4.2 Learning states as user intentions ................................ 47
    4.2.1 Hidden topic Markov model for dialogues .................. 47
    4.2.2 Learning intentions from SACTI-1 dialogues ............... 51
  4.3 Learning the transition model .................................... 54
  4.4 Learning observations and observation model .................... 56
    4.4.1 Keyword observation model .................................. 57
    4.4.2 Intention observation model .................................. 57
  4.5 Example on SACTI dialogues ....................................... 59
    4.5.1 HTMM evaluation ................................................. 62
    4.5.2 Learned POMDP evaluation ................................... 63
  4.6 Conclusions .......................................................... 65

5 Reward model learning .................................................. 68
  5.1 Introduction .......................................................... 68
  5.2 Inverse reinforcement learning in the MDP framework ............ 70
  5.3 Inverse reinforcement learning in the POMDP framework ......... 76
    5.3.1 POMDP-IRL-BT .................................................. 76
    5.3.2 PB-POMDP-IRL .................................................. 82
    5.3.3 PB-POMDP-IRL evaluation ..................................... 85
  5.4 Related work .......................................................... 85
  5.5 POMDP-IRL-MC ........................................................ 87
  5.6 POMDP-IRL-BT and PB-POMDP-IRL performance ..................... 88
  5.7 Conclusions .......................................................... 89

6 Application on healthcare dialogue management ...................... 91
  6.1 Introduction .......................................................... 91
  6.2 Dialogue POMDP model learning for SmartWheeler ................. 94
    6.2.1 Observation model learning .................................... 97
6.2.2 Comparison of the intention POMDP to the keyword POMDP . 99
6.3 Reward model learning for SmartWheeler . . . . . . . . . . . . . . . . . 100
  6.3.1 Choice of features . . . . . . . . . . . . . . . . . . . . . . . . . . 100
  6.3.2 MDP-IRL learned rewards . . . . . . . . . . . . . . . . . . . . . . . 102
  6.3.3 POMDP-IRL-BT evaluation . . . . . . . . . . . . . . . . . . . . . . . 102
  6.3.4 Comparison of POMDP-IRL-BT to POMDP-IRL-MC . . . . . . . . 104
    6.3.4.1 Evaluation of the quality of the learned rewards . . . . . . 105
    6.3.4.2 Evaluation of the spent CPU time . . . . . . . . . . . 107
6.4 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 110

7 Conclusions and future work 111
  7.1 Thesis summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 111
  7.2 Future work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 113

A IRL 115
  A.1 IRL, an ill-posed problem . . . . . . . . . . . . . . . . . . . . . . . . . 115
  A.2 LSPI-IRL . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117
    A.2.1 Choice of features . . . . . . . . . . . . . . . . . . . . . . . . . . 121
    A.2.2 Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 122

Bibliography 123
List of Figures

2.1 The Dirichlet distribution for different values of the concentration parameter, taken from Huang [2005]. ........................................... 12
2.2 Latent Dirichlet allocation. .................................................. 16
2.3 (a): Unigram model (b): Mixture of unigrams (c): Probabilistic latent semantic analysis (PLSA). .................................................. 18
2.4 The hidden Markov model, the shaded nodes are observations \((o_i)\) used to capture hidden states \((s_i)\). ........................................... 20
3.1 The cycle of interaction between an agent and the environment. .... 24
3.2 A 3-step conditional plan of a POMDP with 2 actions and 2 observations. Each node is labeled with an action and each non-leaf node has exactly \(|O|\) observations. .................................................. 33
3.3 The architecture of a spoken dialogue system, adapted from Williams [2006]. ................................................................. 39
3.4 Simulation-based RL: Learning a stochastic simulated dialogue environment from data [Rieser and Lemon, 2011]. ......................... 41
4.1 Hidden states are learned based on an unsupervised learning (UL) method that considers the Markovian property of states between \(n\) and \(n+1\) time steps. Hidden states are represented in the light circles. .... 47
4.2 The HTMM model adapted from Gruber et al. [2007], the shaded nodes are words \((w)\) used to capture intentions \((z)\). ......................... 49
4.3 The maximum likelihood transition model is learned using the extracted actions, \(a\), represented using the shaded square, and the learned states, \(s\), represented in the light circles. ................................. 55
4.4 The observations, \(o\), are learned based on an unsupervised learning (UL) method, and are represented using the shaded circles. ........ 56
4.5 Perplexity trend with respect to increase of the number of observed user utterances. .................................................. 63
4.6 Log likelihood of observations in HTMM as a function of the number of iterations. .................................................. 64
4.7 Average rewards accumulated by the learned dialogue POMDPs with respect to the noise level. ................................. 65
4.8 Average rewards accumulated by the learned dialogue POMDPs with respect to the size of expert dialogues as training data. 66

5.1 The cycle of acting/learning between the agent and environment. The circles represent the models. The model denoted by POMDP includes the POMDP model components, without a reward model, learned from introduced methods in Chapter 4. The learned POMDP model together with action/observation trajectories are used in IRL to learn the reward model denoted by R. The learned POMDP and reward model are used in the POMDP solver to learn/update the policy. 69

5.2 POMDP-IRL-BT illustration example. 79

6.1 The SmartWheeler robot platform. 92

6.2 Comparison of the POMDP-IRL algorithms using keyword features on the learned dialogue POMDP from SmartWheeler. Top: percentage of matched actions. Bottom: sampled value of the learned policy. 106

6.3 Comparison of the POMDP-IRL algorithms using state-action-wise features on the learned dialogue POMDP from SmartWheeler. Top: percentage of matched actions. Bottom: sampled value of learned policy. 108

6.4 Spent CPU time by POMDP-IRL algorithms on SmartWheeler, as the number of expert trajectories (training data) increases. 109

A.1 Number of mismatched actions between the learned policies and the expert policy. 117
List of Tables

1.1 A sample from the SACTI-2 dialogues [Weilhammer et al., 2004]. . . . 2
2.1 Dirichlet distribution example: the results of throwing a die 100 times. 13
2.2 Dirichlet distribution example: the updated posterior probabilities. . . 13
2.3 Dirichlet distribution example: the updated hyper parameters. . . . . 13
2.4 Dirichlet distribution example: the expected value of hyper parameters. 13
2.5 The LDA example: the given text. . . . . . . . . . . . . . . . . . . . 14
2.6 The LDA example: the learned topics. . . . . . . . . . . . . . . . . . . 15
2.7 The LDA example: the topic assignments to the text. . . . . . . . . . 15
3.1 The process of policy learning in the Q-learning algorithm [Schatzmann et al., 2006]. . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
4.1 A sample from the SACTI-1 dialogues [Williams and Young, 2005]. . . 52
4.2 The learned user intentions from the SACTI-1 dialogues. . . . . . . . 53
4.3 Learned probabilities of intentions for the recognized utterances in the SACTI-1 example. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
4.4 Results of applying the two observation models on the SACTI-1 sample. 61
4.5 A sample from SACTI-1 dialogue POMDP simulation. . . . . . . . . 67
5.1 Number of matches for hand-crafted reward POMDPs, and learned re- ward POMDPs, w.r.t. 1415 human expert actions. . . . . . . . . . . . 85
5.2 The learned SACTI-1 specification for IRL experiments. . . . . . . . 88
5.3 POMDP-IRL-BT and PB-POMDP-IRL results on the learned POMDP from SACTI-1: Number of matched actions to the expert actions. . . 89
6.1 A sample from the SmartWheeler dialogues [Pineau et al., 2011]. . . . 93
6.2 The list of the possible actions, performed by SmartWheeler. . . . . . 94
6.3 The learned user intentions from the SmartWheeler dialogues. . . . . 95
6.4 A sample from the results of applying HTMM on SmartWheeler. . . . 96
6.5 The SmartWheeler learned states. . . . . . . . . . . . . . . . . . . . . 97
6.6 A sample from the results of applying the two observation models on the SmartWheeler dialogues. . . . . . . . . . . . . . . . . . . . . 98
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7</td>
<td>The performance of the intention POMDP vs. the keyword POMDP, learned from the SmartWheeler dialogues.</td>
<td>100</td>
</tr>
<tr>
<td>6.8</td>
<td>Keyword features for the SmartWheeler dialogues.</td>
<td>101</td>
</tr>
<tr>
<td>6.9</td>
<td>Top: The assumed expert reward model for the dialogue MDP/POMDP learned from SmartWheeler dialogues. Bottom: The learned reward model for the learned dialogue MDP from SmartWheeler dialogues using keyword features.</td>
<td>103</td>
</tr>
<tr>
<td>6.10</td>
<td>The policy of the learned dialogue MDP from SmartWheeler dialogues with the assumed expert reward model.</td>
<td>104</td>
</tr>
<tr>
<td>A.1</td>
<td>The LSPI-IRL performance using three different features.</td>
<td>122</td>
</tr>
</tbody>
</table>
List of Algorithms

1. The descriptive algorithm to learn the dialogue POMDP model components using unannotated dialogues. .................................................. 5

2. The policy iteration algorithm for MDPs. .................................................. 31

3. The value iteration algorithm for MDPs. .................................................. 32

4. The value iteration algorithm in POMDPs adapted from Williams [2006]. 35

5. Point-based value iteration algorithm for POMDPs adapted from Williams [2006]. .................................................. 37

6. The HTMM generative model, adapted from Gruber et al. [2007]. ........ 50

7. MDP-IRL: inverse reinforcement learning in the MDP framework, adapted from [Ng and Russell, 2000]. .................................................. 74

8. POMDP-IRL-BT: inverse reinforcement learning in the POMDP framework using belief transition estimation. .................................................. 81

9. Point-based POMDP-IRL: a point-based algorithm for IRL in the POMDP framework. .................................................. 84

10. LSPI-IRL: inverse reinforcement learning using LSPI for estimating the policy of the candidate rewards. .................................................. 118
Notation and acronyms

The following basic notation is used in this thesis:

- **x**: Bold lower-case letters represent vectors
- **X**: Bold upper-case letters represent matrices
- **a**: Italic letters refer to scalar values
- **x ← a**: Assignment of x to value of a
- **Pr(s)**: The discrete probability of event s, the probability mass function
- **p(x)**: The probability density function for a continuous variable x
- **{a_1, \ldots, a_n}**: A finite set defined by the elements composing the set
- **N**: Number of intentions
- **K**: Number of features
- **B**: Number of trajectories

Commonly-used acronyms include:

- **ASR**: Automatic speech recognition
- **IRL**: Inverse reinforcement learning
- **MDP**: Markov decision Process
- **PBVI**: Point-based value iteration
- **POMDP**: Partially observable Markov decision process
- **RL**: Reinforcement learning
- **SDS**: Spoken dialogue systems

Acronyms defined in this thesis include:

- **MDP-IRL**: IRL in the MDP framework
- **POMDP-IRL**: IRL in the POMDP framework
- **PB-POMDP-IRL**: Point-based POMDP-IRL
- **POMDP-IRL-BT**: POMDP-IRL using belief transition estimation
- **POMDP-IRL-MC**: POMDP-IRL using Monte Carlo estimation
Chapter 1

Introduction

Spoken dialogue systems (SDSs) are the systems that help the human user to accomplish a task using the spoken language. For example, users can use an SDS to get information about bus schedules over the phone or internet, to get information about a tourist town, to command a wheelchair to navigate in an environment, to control a music player in an automobile, to get information from customer care to troubleshoot devices, and many other tasks. Building SDSs is a difficult problem since automatic speech recognition (ASR) and natural language understanding (NLU) make errors which are the sources of uncertainty in SDSs. In addition, the human user behavior is not completely predictable. The users may change their intentions during the dialogue, which makes the SDS environment stochastic.

Consider the example in Table 1.1 taken from SACTI-2 data set of dialogues [Weilhammer et al., 2004], where SACTI stands for simulated ASR-channel: tourist information. The first line of the table shows the user utterance, $u_1$. Because of the ASR errors, this utterance is recognized by the machine as the line in the braces, i.e., $\tilde{u}_1$. The next line, $m_1$, shows the machine’s response to the user. For each dialogue utterance, the machine’s goal is first to capture the user intention and then to execute the best action that satisfies the user intention. For instance, consider the received user utterance in the first dialogue turn: $\tilde{u}_1$ : [Is there a good restaurant week an hour tonight]. In this turn, the machine can predict the user intention as information request for food places since the utterance contains the only keyword restaurant. However, it is not the case for the received user utterance in the second turn: $\tilde{u}_2$ : [No I think late like uh museum price restaurant]. This utterance contains misleading words such as museum that can be a strong observation for other user intentions, such as user intention for visiting areas. Ideally, in $\tilde{u}_2$ the machine should estimate that the user is looking for food places, and consider museum as a corrupted word, since the user has been asking for restaurant in his previous utterance, $\tilde{u}_1$. It may however consider
Table 1.1: A sample from the SACTI-2 dialogues [Weilhammer et al., 2004].

| $u_1$ | Is there a good restaurant we can go to tonight |
| $\tilde{u}_1$ | [Is there a good restaurant week an hour tonight] |
| $m_1$ | Would you like an expensive restaurant |

| $u_2$ | No I think we’d like a medium priced restaurant |
| $\tilde{u}_2$ | [No I think late like uh museum price restaurant] |
| $m_2$ | Cheapest restaurant is eight pounds per person |

| $u_3$ | Can you tell me the name |
| $\tilde{u}_3$ | [Can you tell me the name] |
| $m_3$ | bochka |

| $u_4$ | Thank you can you show me on the map where it is |
| $\tilde{u}_4$ | [Thank you can you show me i’m there now where it is] |
| $m_4$ | It’s here |

| $u_5$ | Thank you |
| $\tilde{u}_5$ | [Thank u] |

museum as a change of the user intention, and mis-estimate that the user is looking for visiting areas.

Similarly, the machine has to resolve another problem in the received utterance in the third turn: $\tilde{u}_3$: [Can you tell me the name]. Here, there is no keyword restaurant, however, the machine needs to estimate that the user is actually requesting information for food places basically because the user has been asking about food places in the previous utterances.

In addition, the natural language understanding is challenging. For instance, there are several ways of expressing an intention. This is notable for instance in SmartWheeler, which is an intelligent wheelchair to help persons with disabilities. SmartWheeler is equipped with an SDS, thus the users can give their commands through the spoken language besides a joystick. The users may say a command in different ways. For instance for turning right, the user may say:

- turn right a little please,
• turn right,
• right a little,
• right.

And many other ways to say the same intentions. As a response, SmartWheeler can perform the TURN RIGHT A LITTLE action or ask for REPEAT.

Such problems become more challenging when the user utterance is corrupted by ASR. For instance, SmartWheeler may need to estimate that the user asks for turn right from the ASR output, 10 writer little. We call domains such as SmartWheeler intention-based dialogue domains. In such domains, the user intention is the dialogue state which should be estimated by the machine to be able to perform the best action.

In this context, performing the best action in each dialogue state (or the estimated dialogue state) is a challenging task due to the uncertainty introduced by ASR errors and NLU problems as well as the stochastic environment made by user behavior change. In stochastic domains where the decision making is sequential, the suitable formal framework is the Markov decision process (MDP). However, the MDP framework considers the environment as fully observable and this does not conform to real applications which are partially observable such as SDSs. In this context, the partially observable MDP (POMDP) framework can deal this constraint of uncertainty.

In fact, the POMDP framework has been used to model the uncertainty and stochasticity of SDSs in a principled way [Roy et al., 2000; Zhang et al., 2001a,b; Williams and Young, 2007; Thomson, 2009; Gašić, 2011]. The POMDP framework is an optimization framework that supports automated policy solving by optimizing a reward model, while considers the states partially observable. In this framework, the reward model is the crucial model component that directly affects the optimized policy and is a major topic of this thesis, and is discussed further in this section. The optimized policy depends also on other components of the POMDP framework. The POMDP framework includes model components such as: a set of states, a set of actions, a set of observations, a transition model, an observation model, a reward model, etc.

For the example shown in Table 1.1, if we model the control module as a dialogue POMDP, the POMDP states can be considered as the possible user intentions [Roy et al., 2000], i.e., the user information need for food places, visit areas, etc. The POMDP actions include \(m_1, m_2, \ldots\), and the POMDP observations are the ASR output utterances, i.e., \(\tilde{u}_1, \tilde{u}_2, \ldots\), or the keywords extracted from the ASR output utterances. At any case, the observations provide only partial information about the POMDP states, i.e., the user intentions.

The transition model is a probability model representing stochasticity in the domain
and it needs to be learned from the dialogues. For example, the transition model can encode the probability that the user changes his intention between the dialogue turns after receiving the machine’s action. The observation model is a probability model for uncertainty in the domain. For instance, the probability that a particular keyword represents a particular state, say the probability that the keyword restaurant leads to the state food places.

The POMDP reward model encodes the immediate reward for the machine’s executing an action in a state. The reward model which can also be considered as a cost function is the most succinct element that encodes the performance of the machine. For example, in the dialogue POMDPs the reward model is usually defined as: (i) a small negative number (for instance -1) for each action of the machine at any dialogue turn, (ii) a large positive reward (for instance +10) if the dialogue ends successfully, and (iii) a large negative reward (for instance -100) otherwise.

Given a POMDP model, we can apply dynamic programming techniques to solve the POMDP, i.e., to find the (near) optimal policy [Cassandra et al., 1995]. The optimal policy is the policy that optimizes the reward model for any dialogue state sequence. The POMDP’s (near) optimal policy, shortly called the POMDP policy, represents the dialogue manager’s strategy for any dialogue situation. That is, the dialogue manager performs the best action at any dialogue state based on the optimized policy.

Estimating the POMDP model components is a significant issue; as the POMDP model has direct impact on the POMDP policy and consequently on the applicability of the POMDP in the domain of interest. In this context, the SDS researchers in both academia and industry have addressed several practical challenges of applying POMDPs to SDS [Roy et al., 2000; Williams, 2006; Paek and Pieraccini, 2008]. In particular, learning the SDS dynamics ideally from the available unannotated and noisy dialogues is a challenge for us.

In many real applications including SDSs, it is usual to have large amount of unannotated data, such as web-based spoken query retrieval [Ko and Seo, 2004]. Manually annotating the data is an expensive task, thus learning from unannotated data is an interesting challenge which is tackled using unsupervised learning methods. Therefore, we are interested in learning the POMDP model components based on the available unannotated data.

POMDPs, unlike MDPs, have scalability issues. That is, finding the (near) optimal policy of the POMDP highly depends on the number of states, actions and observations. In particular, the number of observations can exponentially increase the number of conditional plans [Kaelbling et al., 1998]. For example, in most non-trivial dialogue domains, the POMDP model can include hundreds or thousands of observations such as words or user utterances. In the example given in Table 1.1, \( \tilde{u}_1, \tilde{u}_2, \tilde{u}_3, \) and \( \tilde{u}_4 \), together
with many other possible utterances, can be considered as observations. Finding the optimal policy of such a POMDP is basically intractable.

Finally, as mentioned above, the reward model of a POMDP highly affects the optimized policy. The reward model is perhaps the most hand-crafted aspect of the optimization frameworks such as POMDPs [Paek and Pieraccini, 2008]. Using Inverse Reinforcement Learning (IRL) [Ng and Russell, 2000], a reward model can be determined from behavioral observation. Fortunately, learning the reward model using IRL methods have already been proposed for the general POMDP framework [Choi and Kim, 2011], paving the way for investigating its use for dialogue POMDPs.

1.1 Approach

In this thesis, we propose methods for learning the dialogue POMDP model components from unannotated and noisy dialogues of intention-based dialogue domains. The big picture of this thesis is presented in the descriptive Algorithm 1. The input to the algorithm is any unannotated dialogue set. In this paper, we use SACTI-1 dialogue data [Williams and Young, 2005] and SmartWheeler dialogues [Pineau et al., 2011].

In step 1, we address learning the dialogue intentions from unannotated dialogues using an unsupervised topic modeling approach, and make use of them as the dialogue POMDP states. In step 2, we directly extract the actions from the dialogue set and learn a maximum likelihood transition model using the learned states. In step 3, we reduce observations significantly and learn the observation model. Specifically, we propose two observation models: the keyword model and the intention model.

Building on the learned dialogue POMDP model components, we propose two IRL algorithms for learning the dialogue POMDP reward model from dialogues, in step 4. The learned reward model makes the dialogue POMDP model complete, which can be used in an available model-based POMDP solver to find the optimal policy.

In this thesis, we present several illustrative examples. We use SACTI-1 dialogues to run the proposed methods and show the results throughout the thesis. In the end, we apply the proposed methods on healthcare dialogue management in order to learn a dialogue POMDP from dialogues collected by an intelligent wheelchair, called SmartWheeler.

\footnote{Note that the proposed methods of this thesis have been applied on both dialogue sets, SACTI-1 and SmartWheeler. But, for historical reasons, methods of step 1 and step 2 in the descriptive Algorithm 1 have been mostly evaluated on SACTI-1, whereas methods of step 3 and step 4 have been mostly evaluated on SmartWheeler.}
Algorithm 1: The descriptive algorithm to learn the dialogue POMDP model components using unannotated dialogues.

**Input:** The unannotated dialogue set of interest

**Output:** The learned dialogue POMDP model components that can be used in a POMDP solver to find the (near) optimal policy

1. Learn the dialogue intentions from unannotated dialogues using an unsupervised topic modeling approach, and make use of them as the dialogue POMDP states;
2. Extract actions directly from dialogues and learn a maximum likelihood transition model using the learned states;
3. Reduce observations significantly and learn the observation model;
4. Learn the reward model based on the IRL technique and using the learned POMDP model components;

1.2 Main contributions

This thesis includes the contributions which have been published in international conferences [Chinaei et al., 2009; Boularias et al., 2010; Chinaei and Chaib-draa, 2012] as well as Canadian conferences [Chinaei and Chaib-draa, 2011; Chinaei et al., 2012]. In this section, we briefly describe our contributions, state to which step in the descriptive Algorithm 1 each of them belongs, and in which chapter each is explained in detail.

Learning user intentions from data for dialogue POMDP states (Chapter 4): This contribution is with respect to step 1 in the descriptive Algorithm 1, i.e., learning the states based on an unsupervised learning method. In this contribution, we propose to learn the states by learning the user intentions occurred in the dialogue set using a topic modeling approach, Hidden Topic Markov Model (HTMM) [Gruber et al., 2007]. HTMM is a variation of Latent Dirichlet Allocation (LDA) which considers Dirichlet distribution for generating the topics in text documents [Blei et al., 2003]. HTMM adds Markovian assumption to LDA to be able to exploit the Markovian property between sentences in the documents. Thus, HTMM can be seen both as a variation of HMM (Hidden Markov Model) and a variation of LDA. In this contribution, we adapt HTMM so that we can learn user intentions from the dialogue set. Our experimental results show that HTMM learns proper user intentions that can be used as dialogue states, and is able to exploit the Markovian property between dialogue utterances adequately.

This contribution resulted to our first publication in the SDS domain, which also received the best student paper award in an international artificial intelligence...
conference [Chinaei et al., 2009]. Moreover, a version of the paper has been included in the Communications in Computer and Information Science (CCIS) series published by Springer.

**Learning dialogue POMDP models from data including a maximum likelihood transition model (Chapter 4):** For step 2 of the descriptive Algorithm 1, we use the learned user intentions as the dialogue POMDP states, and learn a maximum likelihood transition model using the extracted actions from the dialogue set. The learned transition model estimates the chance of user intention change in dialogue turns, i.e., the estimate of user behavior stochasticity.

In this contribution we also learn the observation model from data and apply the methods on SACTI-1 dialogues to learn a dialogue POMDP. Our experimental results show that the quality of the learned models increases by increasing the number of dialogues as training data. Moreover, the experiments based on simulation show that the introduced method is robust to the ASR noise level. These results have been published in Chinaei and Chaib-draa [2011].

**Learning observation models from data (Chapter 4):** This contribution is about step 3 in the descriptive algorithm 1, i.e., reducing the observations significantly and learn an observation model. We propose two crisp observation sets and their subsequent observation models from real dialogues, namely keyword observations and intention observations. The keyword observation model is learned using a maximum likelihood method. On the other hand, the intention observation model is learned by exploiting the learned intentions from the dialogue set, the learned intention model for each dialogue, and the learned conditional model of observations and words from the set of dialogues. For instance for the first ASR output in Table 1.1, the keyword model uses the keyword *restaurant* as an observation. However, the intention model uses the underlying intention *food* places as an observation. Based on experiments on two dialogue domains, we observe that the intention observation model performance is substantially higher than the keyword model one. This contribution has been published in Chinaei et al. [2012].

**Learning reward models using expert trajectories and the proposed POMDP-IRL algorithm 1 (Chapter 5):** This contribution is about step 4 in the descriptive Algorithm 1, where we propose to learn the reward model based on IRL and using the learned POMDP model components. Specifically, we propose algorithms for learning the reward model of POMDPs from data. In IRL techniques a reward model (or a cost function) is learned from an (assumed) expert. In SDS, the expert is either the dialogue manager of the SDS, which has performed the machine’s actions in dialogues, or a human who has performed the actions by
playing the role of a dialogue manager (in a Wizard-of-Oz setting).

We first propose an IRL algorithm in POMDP framework which is called POMDP-IRL-BT (BT for belief transition). The POMDP-IRL-BT algorithm works on the expert belief states available in the dialogues by approximating a belief transition model similar to the MDP transition models. Finally, the POMDP-IRL-BT algorithm approximates the reward model of the expert iteratively by maximizing the sum of the margin between the expert policy and other policies. Moreover, we implement the Monte-Carlo estimator in the POMDP-IRL-BT algorithm to make the POMDP-IRL-MC algorithm (MC for the Monte Carlo). The POMDP-IRL-MC algorithm estimates the policy values using Monte Carlo estimator rather than by estimating the belief transition. Then, we compare POMDP-IRL-BT to POMDP-IRL-MC. Our experimental results show that POMDP-IRL-BT outperforms POMDP-IRL-MC. However, POMDP-IRL-MC does scale better than POMDP-IRL-BT. This contribution with its application on SmartWheeler dialogues have been published in Chinaei and Chaib-draa [2012].

Learning reward models using expert trajectories and the proposed POMDP-IRL algorithms 2 (Chapter 5): We also propose a point-based POMDP-IRL algorithm, called PB-POMDP-IRL, that approximates the value of the new beliefs that occurs in the computation of the policy values using the expert beliefs in the expert trajectories. This algorithm is compared to POMDP-IRL-BT based on experiments on the learned dialogue POMDP from SACTI-1 dialogues. The results show that POMDP-IRL-BT learns reward models that accounts for the expert policy better than the reward models learned by PB-POMDP-IRL. The PB-POMDP-IRL algorithm with its application on SACTI-1 dialogues has been published in Boularias et al. [2010].

In addition to the above mentioned contributions, to the best of our knowledge, this is the first work that proposes and implements an end-to-end learning approach for dialogue POMDP model components. That is, starting from scratch, it learns the state, the transition model, the observation and the observation model and finally the reward model. These altogether form a significant set of contributions that can potentially inspire substantial further work.

1.3 Thesis structure

The rest of the thesis is organized as follows. We describe the necessary background knowledge in Chapter 2. In particular, we introduce the probability theory, Dirichlet distributions, MDP and POMDP frameworks. In Chapter 4 we go through steps 1 to 3


in the descriptive Algorithm 1. That is, we propose the methods for learning more basic dialogue POMDP model components: the states and transition model, the observations and observation model. Then in Chapter 5, we review inverse reinforcement learning (IRL) in the MDP framework followed by our proposed POMDP-IRL algorithms for learning dialogue POMDP reward model. In Chapter 6, we apply the whole methods on SmartWheeler, to learn a dialogue POMDP from SmartWheeler dialogues. Finally, we conclude and address the future work in Chapter 7.
Chapter 2

Topic modeling

Topic modeling techniques are used to discover the topics for (unlabeled) texts. As such, they are considered as unsupervised learning techniques which try to learn the patterns inside the text by considering words as observations. In this context, latent Dirichlet allocation (LDA) is a Bayesian topic modeling approach which has useful properties particularly for practical applications [Blei et al., 2003]. In this section, we go through LDA by first reviewing the Dirichlet distribution, which is the basic distribution used in LDA.

2.1 Dirichlet distribution

Dirichlet distribution is the \textit{conjugate} prior for multinomial distribution likelihood [Kotz et al., 2000; Balakrishnan and Nevzorov, 2003; Fox, 2009]. Specifically, the conjugate prior of a distribution has the property that after updating the prior, the posterior also has the same functional form as the prior [Hazewinkel, 2002; Robert and Casella, 2005]. It has been shown that conjugate priors are found only inside the exponential families [Brown, 1986].

2.1.1 Exponential distributions

The density function of exponential distributions has a factor called \textit{sufficient statistic}. The sufficient statistic is the sufficient function of the sample data (as reflected by its name) such that no other statistic that can be calculated from the sample data provides any additional information than the sufficient statistic [Fisher, 1922; Hazewinkel, 2002]. For instance, the maximum likelihood estimator in exponential families depends on the sufficient statistic but not all of observations.
Chapter 2. Topic modeling

The exponential families have the property that the dimension of sufficient statistic is bounded even if the size of observations goes to infinity, except a few member of exponential families such as uniform distribution. Moreover, the important property of exponential families is inside the theorems independently proved by Pitman [1936], Koopman [1936], and Darmois [1935] approximately at the same time. This property leads to efficient parameter estimation methods in exponential families. Examples of exponential families are the normal, Gamma, Poison, multinomial, and Dirichlet distributions. In particular, the Dirichlet distribution is the conjugate prior for the multinomial distribution likelihood.

2.1.2 Multinomial distribution

For the multinomial distribution, consider the trial of \( n \) events with observations \( \mathbf{y} = (y_1, \ldots, y_n) \) and the parameters \( \mathbf{\pi} = (\pi_1, \ldots, \pi_k) \) where the observation of each event can take \( K \) possible values. For instance, in events of rolling a fair die \( n \) times, each observation \( y_i \) can take \( K = 6 \) values with equal probabilities, \( (\pi_1 = \frac{1}{6}, \ldots, \pi_k = \frac{1}{6}) \). Under such condition, this experiment is governed by a multinomial distribution. Formally, for the probability of having an observation \( \mathbf{y} = (y_1, \ldots, y_n) \) given the parameters \( \mathbf{\pi} = (\pi_1, \ldots, \pi_k) \) we have:

\[
p(y|\mathbf{\pi}) = \frac{n!}{\prod_{i=1}^{K} n_i!} \prod_{i=1}^{K} \pi_i^{n_i}
\]

where

\[
n_i = \sum_{j=1}^{n} \delta(y_j, i)
\]

in which \( \delta(x, y) \) is the Kronecker delta function; \( \delta(x, y) = 1 \) if \( x = y \), and zero otherwise. Moreover, it can be shown that in multinomial distribution, the expectation of number of times that the value \( i \) is observed over \( n \) trials is:

\[
E(Y_i) = n\pi_i
\]

and its variance is:

\[
Var(Y_i) = n\pi_i(1 - \pi_i)
\]

2.1.3 Dirichlet distribution

For the conjugate prior of the likelihood of multinomial distribution, i.e., \( p(\mathbf{\pi}|\mathbf{y}) \), assume that the prior \( p(\mathbf{\pi} = (\pi_1, \ldots, \pi_k)) \) is drawn from Dirichlet distribution with the hyper parameters \( \mathbf{\alpha} = (\alpha_1, \ldots, \alpha_k) \) then the posterior \( p(\mathbf{\pi}|\mathbf{y}) \) is also drawn from Dirichlet
distribution with the hyper parameters \((\alpha_1 + n_1, \ldots, \alpha_k + n_k)\). Recall that \(n_i\) is the number of times the value \(i\) has been observed in the last trial, where \(1 \leq i \leq K\).

This is the useful property of Dirichlet distribution which says that for updating the prior to get the posterior it suffices only to update the hyper parameters. That is, having a Dirichlet prior with the hyper parameters \(\alpha = (\alpha_1, \ldots, \alpha_k)\), after observing observations \((n_1, \ldots, n_k)\) the posterior hyper parameters become \((\alpha_1 + n_1, \ldots, \alpha_k + n_k)\). This property is discussed in the illustrative example further in this section.

Then, Dirichlet distribution for the parameter \(\pi\) with hyper parameter \(\alpha\) would be:

\[
p(\pi|\alpha) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \prod_i \pi_i^{\alpha_i - 1}
\]

where \(\Gamma(x)\) is the standard Gamma function. Note that Gamma function is an extension of factorial function. That is, for positive numbers Gamma function is the factorial function, i.e., \(\Gamma(n) = n!\). Moreover, it can be shown that the expectation of Dirichlet prior \(\pi\) is:

\[
E(\pi_i) = \frac{\alpha_i}{s} \tag{2.1}
\]

and its variance is:

\[
Var(\pi_i) = \frac{E(\pi_i)(1 - E(\pi_i))}{s + 1}
\]

where \(s = \alpha_1 + \ldots + \alpha_k\) and is called the concentration parameter. The concentration parameter controls how concentrated the distribution is around its expected value [Sudderth, 2006]. The higher \(s\) is, the lower is the variance of the parameters. Moreover, given the concentration parameter \(s\), the higher the hyper \(\alpha_i\) is, the higher the expected value of \(\pi_i\) is. Therefore, the Dirichlet hyper parameters \(\alpha = (\alpha_1, \ldots, \alpha_i)\) operate as a confidence measure.

Figure 2.1 plots 3 Dirichlet distributions with 3 values for \(s\) in three unit simplex (with 3 vertices). Note that \(p(\pi)\) is a point in each simplex and \(0 \leq \pi_i\), and \(\sum^K \pi_i = 1\). Figure 2.1 shows that the higher the \(s\) is, the more concentration is around its expected value. In addition, the simplex in the middle has a high \(s\) whereas the one in the right has a lower \(s\).

Neapolitan [2004] proved the useful property for the posterior of Dirichlet distribution. Suppose we are about to repeatedly perform an experiment with \(k\) outcomes \(x_1, x_2, \ldots, x_k\). We assume exchangeable observations and present our prior belief concerning the probability of heads using a Dirichlet distribution with the parameters \(\alpha = (\alpha_1, \ldots, \alpha_k)\). Then, our prior probabilities become:

\[
p(x_1) = \frac{\alpha_1}{m} \quad \ldots \quad p(x_k) = \frac{\alpha_k}{m}
\]
Chapter 2. Topic modeling

Figure 2.1: The Dirichlet distribution for different values of the concentration parameter, taken from Huang [2005].

where \( m = \alpha_1 + \ldots + \alpha_k \).

After observing \( x_1, \ldots, x_k \) occurs respectively \( n_1, \ldots, n_k \) times in \( n \) trials where \( n = n_1 + \ldots + n_k \). Then, our posterior probabilities become as follows:

\[
p(x_1|n_1, \ldots, n_k) = \frac{\alpha_1 + n_1}{s = m + n}
\]

\[
\ldots
\]

\[
p(x_k|n_1, \ldots, n_k) = \frac{\alpha_k + n_k}{s = m + n}
\]

2.1.4 Example on the Dirichlet distribution

Here, we present an illustrative example for the Dirichlet distribution, taken from Neapolitan [2009]. Suppose we have an asymmetrical, six-sided die, and we have little idea of the probability of each side coming up. However, it seems that all sides are equally likely. So, we assign equal initial confidence about observing each number 1 to 6 appear by the die on the Dirichlet hyper parameters \( \alpha = (\alpha_1, \ldots, \alpha_k) \) as follows:

\[
\alpha_1 = \alpha_2, \ldots, \alpha_6 = 3
\]

Then, we have \( s = 3 \times 6 = 18 \), and the prior probabilities are as follows:

\[
p(1) = p(2) = \ldots = p(6) = \frac{\alpha_i}{s} = \frac{3}{18} = 0.16667
\]

Next, suppose that we throw the die 100 times, with the following results shown in Table 2.1.

Using Equation (2.2), the posterior probabilities can be updated as shown in Table 2.2.
Chapter 2. Topic modeling

<table>
<thead>
<tr>
<th>Outcome ($x_i$)</th>
<th>Number of Occurrences ($n_i$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>27</td>
</tr>
<tr>
<td>$n$</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 2.1: Dirichlet distribution example: the results of throwing a die 100 times.

| $p(i|10, 15, 5, 30, 13, 27)$ | $\frac{\alpha_1 + n_1}{s}$ | $\frac{3 + 10}{18 + 100}$ | 0.110 |
|---------------------------|-----------------------------|---------------------------|------|
| $p(2|10, 15, 5, 30, 13, 27)$ | $\frac{\alpha_1 + n_1}{s}$ | $\frac{3 + 15}{18 + 100}$ | 0.153 |
| $p(3|10, 15, 5, 30, 13, 27)$ | $\frac{\alpha_1 + n_1}{s}$ | $\frac{3 + 5}{18 + 100}$  | 0.067 |
| $p(4|10, 15, 5, 30, 13, 27)$ | $\frac{\alpha_1 + n_1}{s}$ | $\frac{3 + 30}{18 + 100}$ | 0.280 |
| $p(5|10, 15, 5, 30, 13, 27)$ | $\frac{\alpha_1 + n_1}{s}$ | $\frac{3 + 13}{18 + 100}$ | 0.136 |
| $p(6|10, 15, 5, 30, 13, 27)$ | $\frac{\alpha_1 + n_1}{s}$ | $\frac{3 + 27}{18 + 100}$ | 0.254 |

Table 2.2: Dirichlet distribution example: the updated posterior probabilities.

Note in the example that the new value for the concentration parameter becomes $s = m + n$, where $m = 18$ ($\alpha_1 + \ldots + \alpha_k$), and $n = 100$ (the number of observations). Moreover, the new values of hyper parameters become as shown in Table 2.3.

<table>
<thead>
<tr>
<th>$\alpha_1$</th>
<th>$\alpha_1 + n_1$</th>
<th>3 + 10</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_2$</td>
<td>$\alpha_2 + n_2$</td>
<td>3 + 15</td>
<td>18</td>
</tr>
<tr>
<td>$\alpha_3$</td>
<td>$\alpha_3 + n_3$</td>
<td>3 + 5</td>
<td>8</td>
</tr>
<tr>
<td>$\alpha_4$</td>
<td>$\alpha_4 + n_4$</td>
<td>3 + 30</td>
<td>33</td>
</tr>
<tr>
<td>$\alpha_5$</td>
<td>$\alpha_5 + n_2$</td>
<td>3 + 13</td>
<td>16</td>
</tr>
<tr>
<td>$\alpha_6$</td>
<td>$\alpha_6 + n_2$</td>
<td>3 + 27</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 2.3: Dirichlet distribution example: the updated hyper parameters.

Using Equation (2.1), $E(\pi_i) = \alpha_i/s$, the expected value of the parameters can be calculated as shown in Table 2.4.

Comparing the values in Table 2.4 to the ones in Table 2.2, we can see another important property of the Dirichlet distribution. That is, the number of observations directly reveals the confidence on the expected value of parameters.

In this section, we observed the Dirichlet distribution’s useful properties:

1. The Dirichlet distribution is the conjugate prior for likelihood of multinomial
$$E(\pi_1) = \alpha_1/s = 13/118 = 0.110$$
$$E(\pi_2) = \alpha_2/s = 18/118 = 0.153$$
$$E(\pi_3) = \alpha_3/s = 8/118 = 0.280$$
$$E(\pi_4) = \alpha_4/s = 33/118 = 0.067$$
$$E(\pi_5) = \alpha_5/s = 16/118 = 0.136$$
$$E(\pi_6) = \alpha_6/s = 30/118 = 0.254$$

Table 2.4: Dirichlet distribution example: the expected value of hyper parameters.

2. For updating the posterior of multinomial distribution with Dirichlet prior, we need only to update the Dirichlet prior by adding the observation counts to the Dirichlet hyper prior, and

3. The number of observations directly reveals the confidence on the expected value of the parameters.

Because of these important properties, the Dirichlet distribution is applied largely in different applications. In particular, latent Dirichlet allocation (LDA) assumes that the learned parameters follow the Dirichlet distribution. The following section describes the LDA method.

### 2.2 Latent Dirichlet allocation

Latent Dirichlet allocation (LDA) is a latent Bayesian topic model which is used for discovering the hidden topics of documents [Blei et al., 2003]. In this model, a document can be represented as a mixture of the hidden topics, where each hidden topic is represented by a distribution over words occurred in the document. Suppose we have the sentences shown in Table 2.5.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>I eat orange and apple since those are juicy.</td>
</tr>
<tr>
<td>2:</td>
<td>The weather is so windy today.</td>
</tr>
<tr>
<td>3:</td>
<td>The hurricane Catherine passed with no major damage.</td>
</tr>
<tr>
<td>4:</td>
<td>Watermelons here are sweat because of the hot weather.</td>
</tr>
<tr>
<td>5:</td>
<td>Tropical storms usually end by November.</td>
</tr>
</tbody>
</table>

Table 2.5: The LDA example: the given text.

Then, the LDA method automatically discovers the topics that the given text contain. Specifically, given 2 asked topics, LDA can learn the two topics and the topic assignments to the given text. The learned topics are represented using the words and their
probabilities of occurring for each topic as presented in Table 2.6. The topic representation for topic A illustrates that this topic is about *fruits*. And, the topic representation for topic B illustrates that this topic B is about the *weather*. Then, the topic assignment for each sentence can be calculated as presented in Table 2.7.

<table>
<thead>
<tr>
<th>Topic A</th>
<th>Topic B</th>
</tr>
</thead>
<tbody>
<tr>
<td>orange</td>
<td>weather</td>
</tr>
<tr>
<td>apple</td>
<td>windy</td>
</tr>
<tr>
<td>juicy</td>
<td>hot</td>
</tr>
<tr>
<td>sweat</td>
<td>storm</td>
</tr>
<tr>
<td>. . .</td>
<td>. . .</td>
</tr>
</tbody>
</table>

Table 2.6: The LDA example: the learned topics.

| Sentence 1: Topic A 100% Topic B 0% |
| Sentence 2: Topic A 0% Topic B 100% |
| Sentence 3: Topic A 0% Topic B 100% |
| Sentence 4: Topic A 65% Topic B 35% |
| Sentence 5: Topic A 0% Topic B 100% |

Table 2.7: The LDA example: the topic assignments to the text.

Formally, given a document in the form of \( d = (w_1, \ldots, w_M) \) in a document corpus (set), \( D \), and given \( N \) asked topics, the LDA model learns two parameters:

1. The parameter \( \theta \) which is generated from the Dirichlet prior \( \alpha \).
2. The parameter \( \beta \) which is generated from Dirichlet prior \( \eta \).

The first parameter, \( \theta \), is a vector of size \( N \) for distribution of hidden topics, \( z \). The second one, \( \beta \), is a matrix of size \( M \times N \) in which the column \( j \) stores the probability of each word given the topic \( z_j \).

Figure 2.2 shows the LDA model in the plate notation in which the boxes are plates, that represents replicates. The shaded nodes are the observation nodes, i.e., the words \( w \). The unshaded nodes \( z \) represent hidden topics. Then, the generative model of LDA performs as follows:

1. For each document, \( d \), a parameter, \( \theta \), is drawn for the distribution of hidden topics based on multinomial distribution with the Dirichlet parameters \( \alpha \) (cf. Dirichlet distribution in Section 2.1).
2. For each document set $D$, a parameter, $\beta$, is learned for the distribution of words given topics. Given each topic $z$, the vector $\beta_z$ is drawn based on multinomial distribution with the Dirichlet parameters $\eta$.

3. Generate the $j$th word in the document $i$, $w_{i,j}$, as:
   
   (a) Draw a topic $z_{i,j}$ based on the multinomial distribution with the parameter $\theta_i$.
   
   (b) Draw a word based on the multinomial distribution with the parameter $\phi_{z_{i,j}}$.

**Comparison to earlier models**

Blei et al. [2003] compared the LDA model to the related earlier models such as unigrams and mixture of unigrams [Bishop, 2006; Manning and Schütze, 1999], as well as probabilistic latent semantic analysis (PLSA) [Hofmann, 1999]. These three models are represented in Figure 2.3.

Figure 2.3 (a) shows the unigram model. In unigrams, a document $d = (w_1, \ldots, w_n)$ is a mixture of words. So, the probability of having a document $d$ is calculated as:

$$p(d) = \prod_{w_i} p(w_i)$$

Then, in the mixture of unigrams in Figure 2.3 (b), a word $w$ is drawn from a topic $z$ this time. Under this model, a document $d$ is generated by:

![Figure 2.2](image-url)
1. Draw a hidden topic $z$.

2. Draw each word $w$ based on the hidden topic $z$.

As such, in mixture of unigrams the probability of having the document $d$ is calculated as:

$$p(d) = \sum_z p(z) \prod_{w_i} p(w_i|z)$$

Notice that mixture of unigrams assumes that each document $d$ includes only one hidden topic. This assumption is removed in PLSA model shown in Figure 2.3 (c). In PLSA, a distribution $\theta$ is sampled and attached to each observed document for the distribution of hidden topics. Then, the probability of having a document $d = (w_1, \ldots, w_n)$ is calculated as:

$$p(d) = \sum_z p(z|\theta) \prod_{w_i} p(w_i|z)$$

where $\theta$ is the distribution of hidden topics.

Note also that LDA is similar to PLSA in that both LDA and PLSA learn a parameter $\theta$ for the distribution of hidden topics of each document. Then, the probability of having a document $d = (w_1, \ldots, w_n)$ is calculated using:

$$p(d) = \sum_z p(z|\theta) \prod_{w_i} p(w_i|z)$$

where $\theta$ is the distribution of hidden topics.

In contrast to PLSA, in LDA first a parameter $\alpha$ is generated which is used as the Dirichlet prior for the multinomial distribution $\theta$ of topics. In fact, Dirichlet prior can be used as a natural way to assign more probability to the random variables on which we have more confidence. Moreover, use of Dirichlet prior leads to interesting advantages of LDA over PLSA. First, as opposed to PLSA, LDA does not require to visit a document $d$ to sample a parameter $\theta$. But in LDA, the parameter $\theta$ is generated using the Dirichlet parameter $\alpha$. As such, LDA is a well defined generative model of documents which is able to assign probabilities to a previously unseen document of the corpus. Moreover, LDA is not dependent to the size of corpus and does not overfit as opposed to PLSA [Blei et al., 2003].

So, LDA is a topic modeling approach that considers mixture of hidden topics for documents, where documents are seen as bag of words. However, it does not consider the Markovian property among sentences. Later in this thesis, we introduce a variation of LDA that adds the Markovian property to LDA, for the topic transition from one sentence to the following one. In this context, hidden Markov models (HMMs) are used for modeling Markovian property particularly in texts. In the following section, we briefly review HMMs.
2.3 Hidden Markov models

In Markovian domains the current environment’s state depends on the state in the previous time step, similar to finite state machines. In fact, Markov models are generalized models of finite state machines in which the transitions are not deterministic. That is, in Markov models the current environment state depends on the previous state and the probability of landing to the current state, known as the transition probability [Manning and Schütze, 1999].

In hidden Markov models (HMMs) [Rabiner, 1990], as opposed to Markov models, states are not fully observable, but there is the idea of observations which give the current state of the model with only some probability. So, in HMMs there is an observation model besides the transition model. Similar to the Markov models, in HMMs the transition model is used for estimating the current state of the model with some probability, given the previous state. As such, we can state that an HMM with a deterministic observation model is equivalent to a Markov model, and that a Markov model with a deterministic transition model is equivalent to a finite state machine.

Figure 2.4 shows an HMM where hidden states $s_1, \ldots, s_n$ are inside circles and ob-
observations $o_1, \ldots, o_n$ are noted inside the shaded circles. The Markovian property in HMMs states that at each time step the state of the HMM depends on its previous state $p(s_t|s_{t-1})$, and the current observation depends on the current state $p(o_t|s_t)$.

Formally, an HMM is defined as a tuple $(S, O, A, B, \Pi)$:

- $S = s_1, \ldots, s_N$ is a set of $N$ states,
- The transition probability matrix $A$ 
  $A = \begin{pmatrix} a_{11}, \ldots, a_{1n} \\ \vdots \\ a_{n1}, \ldots, a_{nn} \end{pmatrix}$
  Each $a_{ij}$ represents the probability of moving from state $i$ to state $j$, s.t. $\sum_{j=1}^n a_{ij} = 1$,
- $O = o_1o_2\ldots o_T$, is sequence of $T$ observations, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$,
- $B = b_i(o_t)$, is a sequence of observation likelihoods, also called emission probabilities, each expressing the probability of an observation $o_t$ being generated from a state $i$,
- $\Pi$ is the initial probability model which shows the probability that the model starts with each state in $S$.

Then, there are three fundamental questions that we want to answer in HMMs [Jurafsky and Martin, 2009; Manning and Schütze, 1999]:

1. The first problem is to compute the likelihood of a particular observation sequence. Formally, we want to find out:

   Given an HMM, $\lambda = (A, B)$ and an observation sequence $O$, determine the likelihood $Pr(O|\lambda)$.

2. Learning the most likely state sequence given a sequence of observations and the model. This problem is called decoding. This is interesting for instance in part-of-speech tagging where given a set of words as observations we would like to infer about the most probable tags of the words [Church, 1988]. Formally, we want to find out:
Chapter 2. Topic modeling

Figure 2.4: The hidden Markov model, the shaded nodes are observations \((o_i)\) used to capture hidden states \((s_i)\).

Given as input and HMM \(\lambda = (A, B)\), and a sequence of observations \(O = o_1, o_2, \ldots, o_T\), find the most probable sequence of states, i.e., \((s_1, \ldots, s_T)\).

That is, we want to find out the state sequence that best explains the observations.

3. HTMM training, i.e., learning the HMM parameters. Given a sequence of observations what the most probable model parameters are:

\[
\arg\max_{\lambda} p(o_1, \ldots, o_n | \lambda) \tag{2.3}
\]

This problem is called parameter estimation.

Note that there is no analytical solution for the maximization of parameter estimation in Equation (2.3). This problem is tackled with a well known algorithm named as Baum-Welch or Forward-Backward algorithm [Welch, 2003], which is an Expectation Maximization (EM) algorithm.

In fact, EM is a class of algorithms for learning unknown parameters of a model. The basic idea of is to pretend that the parameters of the model are known and then to infer the probability that each observation belongs to each model [Russell and Norvig, 2010]. Then, the model refit to the observations, where each model is fitted to the all observations with each observation is weighted by the probability that it belongs to that model. This process iterates until convergence.

EM algorithms start with a random parameter, and calculate the probability of observations. Then, they observe in the calculations to find which state transitions and observation probabilities have been used most, and increase the probability of those. This process leads to an updated parameter which gives higher probability to the observations. Then, the following two steps are iterated until convergence: calculating the probabilities of observations given a parameter (expectation) and updating the parameter (maximization).
Formally, an EM algorithm works as follows. Assuming the set of parameter $\Theta$, hidden variables $Z$ and observations $X$. First, the function $Q$ is defined as [Dempster et al., 1977]:

$$Q(\Theta|\Theta^t) = E[\log p(X, Z|\Theta) | X, \Theta^t] \quad (2.4)$$

Then, in the expectation and maximization steps the following calculations are performed:

1. Expectation: $Q(\Theta|\Theta^t)$ is computed.

2. Maximization: $\Theta^{t+1} = \arg\max_{\Theta} Q(\Theta|\Theta^t)$

   That is, the parameter $\Theta^{t+1}$ is set to the $\Theta$ that maximizes $Q(\Theta|\Theta^t)$.

For instance, in Baum-Welch algorithm the expectation and maximization steps are as follows:

1. In the expectation the following two calculations are done:
   - Calculating the expected number of times that observation $o$ has been observed from state $s$ for all states and observations, given the current parameter of the model.
   - Calculating the expected number of times that state transitions from state $s_i$ to state $s_j$ is done, given the current parameters of the model.

2. In the maximization step the parameters $A$, $B$, and $\Pi$ are set to the parameters which maximize the expectations above.

More specifically, the Expectation and Maximization step for HMM parameter learning, can be derived as described in Jurafsky and Martin [2009]:

1. Expectation:

   $$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{Pr(O|\gamma)} \quad \forall \ t \text{ and } j$$

   $$\xi_t(i, j) = \frac{\alpha_t(i)a_ij\beta_{t+1}(j)}{\alpha_T(N)} \quad \forall \ t, \ i, \text{ and } j$$

   where $\alpha_t$ is known as forward path probability:

   $$\alpha_t(j) = Pr(o_1, o_2, \ldots, o_t, s_t = j|\lambda)$$

   and $\beta_t(j)$ is known as backward path probability:

   $$\beta_t(i) = Pr(o_{t+1}, o_{t+2}, \ldots, o_T|s_t = i, \lambda)$$
2. Maximization:

\[
\hat{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \xi(i, j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi(i, j)}
\]

\[
\hat{b}_j(\nu_k) = \frac{\sum_{t=1, \text{s.t. } O_t = \nu_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}
\]

In this section, we introduced the basic methods used in topic modeling. In particular, we studied the LDA method and HMMs, the background for hidden topic Markov model (HTMM). The HTMM approach adds Markovian property to the LDA method, and is introduced in Chapter 4. In the following chapter, we introduce the sequential decision making domain and its application on spoken dialogue systems.
Chapter 3

Sequential decision making in spoken dialogue management

This chapter includes two major sections. In Section 3.1, we introduce sequential decision making and study the supporting mathematical framework for it. We describe the Markov decision process (MDP) and the partially observable MDP (POMDP) frameworks, and present the well known algorithms for solving them. In Section 3.2, we introduce spoken dialogue systems. Then, we study the related work of sequential decision making in spoken dialogue management. In particular, we study the related research on application of the POMDP framework for spoken dialogue management. Finally, we review the user modeling techniques that have been used for dialogue POMDPs.

3.1 Sequential decision making

In sequential decision making, an agent needs to take sequential actions, during the interaction with an environment. The agent’s interaction with the environment can be in a stochastic and/or uncertain situation. That is, the effect of the actions is not completely known (in stochastic domains) and observations from the environment provide incomplete or error-prone information (in uncertain domains). As such, sequential decision making under such condition is a challenging problem.

Figure 3.1 shows the cycle of interaction between an agent and its environment. The agent performs an action and receives an observation in return. The observation can be used by the agent, for instance to update its state and reward. The reward works as a reinforcement from the environment that shows how well the agent performed. In sequential decision making, the agent is required to make decision for sequence of states rather than making a one-shot decision. Then, the sequential decision making is
Chapter 3. Sequential decision making in spoken dialogue management

25

Figure 3.1: The cycle of interaction between an agent and the environment.

performed with the objective of maximizing the long term rewards. The sequence of actions is called a policy, and the major question in sequential decision making is how to find a near optimal policy.

In stochastic domains where the decision making is sequential, the suitable formal framework to find the near optimal policy is the Markov decision process (MDP). However, the MDP framework considers the environment as fully observable and this does not conform to real applications which are partially observable such as SDSs. In this context, the partially observable MDP (POMDP) framework can deal this constraint of uncertainty. The MDP/POMDP frameworks are composed of model components which can be used, for instance, for representing the available stochasticity and uncertainty.

If the MDP/POMDP model components are not known in advance, then reinforcement learning (RL) is used to learn the near optimal policy. In fact, RL is a series of techniques in which the agent learns the near optimal policy in the environment based on the agent's own experience [Sutton and Barto, 1998]. The better the agent acts, the more rewards it achieves. Then, the agent aims to maximize its expected rewards over time. Since in RL the model components are usually unknown, RL is called model-free RL; particularly in spoken dialogue community [Rieser and Lemon, 2011].

On the other hand, if the model components of the underlying MDP/POMDP framework are known in advance, then we can solve MDPs/POMDPs, which is a search through the state space for an optimal policy or path to goal using the available planning
algorithms [Bellman, 1957a]. This method is also called model-based RL, particularly in the spoken dialogue community [Rieser and Lemon, 2011].

In this thesis, we are interested in learning the environment dynamics of a dialogue manager in advance and make use of them in the POMDP model components. We then refer to such dialogue manager as dialogue POMDP. Once the dialogue POMDP model components are learned, we can solve the POMDP for the optimal policy using the available planning algorithms. In the following section, we introduce the MDP and POMDP background.

### 3.1.1 Markov decision processes (MDPs)

A Markov decision process (MDP) is a mathematical framework for decision making under uncertainty [Bellman, 1957b]. A MDP is defined as $(S, A, T, R, \gamma, s_0)$ where,

- $S$ is the set of discrete states,
- $A$ is the set of discrete actions,
- $T$ is the transition model which consists of the probabilities of state transitions:
  \[
  T(s, a, s') = Pr(s_{t+1} = s'|s_t = s, a_t = a),
  \]
  where $s$ is the current state and $s'$ is the next state,
- $R(s, a)$ is the reward of taking action $a$ in the state $s$,
- $\gamma$ is the discount factor, a real number between 0 and 1,
- and $s_0$ is an initial state.

Then, a policy is the selection of an action $a$ in a state $s$. That is, the policy $\pi$ maps each state $s$ to an action $a$, i.e., $a = \pi(s)$. In an MDP, the objective is to find an optimal policy $\pi^*$, that maximizes the value function, i.e., the expected discount of future rewards starting from state $s_0$:

\[
V^\pi(s) = E_{s_t \sim T} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) | \pi, s_0 = s \right]
\]

The value function of a policy can also be recursively defined as:
Chapter 3. Sequential decision making in spoken dialogue management

\[
V^\pi(s) = E_{s_t \sim T} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \right] \pi, s_0 = s
\]

\[
= E_{s_t \sim T} \left[ R(s_0, \pi(s_0)) + \sum_{t=1}^{\infty} \gamma^t R(s_t, \pi(s_t)) \right] \pi, s_0 = s
\]

\[
= R(s, \pi(s)) + E_{s_t \sim T} \left[ \sum_{t=1}^{\infty} \gamma^t R(s_t, \pi(s_t)) \right] \pi
\]

\[
= R(s, \pi(s)) + \gamma E_{s_t \sim T} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \right] \pi, s_0 \sim T
\]

\[
= R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s')
\]

The last equation is known as Bellman equation which recursively find the value function, defined as:

\[
V^\pi(s) = \left[ R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s') \right] \tag{3.1}
\]

And the optimal state-value function \( V^* \) can be found by:

\[
V^*(s) = \max_{\pi} V^\pi(s)
\]

\[
= \max_{\pi} \left[ R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s') \right]
\]

We can also define Bellman value function as a function of state and action, \( Q^\pi(s, a) \), which estimates the expected return of taking action \( a \) in a given state \( s \) and policy \( \pi \):

\[
Q^\pi(s, a) = \left[ R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^\pi(s') \right] \tag{3.2}
\]

3.1.2 Partially observable Markov decision processes (POMDPs)

A partially observable Markov decision process (POMDP) is a more generalized framework for planning under uncertainty where the basic assumption is that the states are only partially observable. A POMDP is represented as a tuple \( (S, A, T, \gamma, R, O, \Omega, b_0) \). That is, a POMDP model includes an MDP model and adds:

- \( O \) is the set of observations,
• $\Omega$ is the observation model:
\[
\Omega(a, s', o') = Pr(o'|a, s'),
\]
for the probability of observing $o'$ after taking the action $a$ which resulted in the state $s'$. 

• and $b_0$ is an initial belief over all states.

Since POMDPs consider the environment partially observable, in POMDPs a belief over states is maintained in the run time as opposed to MDPs which consider states fully observable. So, in the run time if the POMDP belief over state $s$ at the current time is $b(s)$, then after taking action $a$ and observing observation $o$ the POMDP belief in the next time for state $s'$ is denoted by $b'(s')$ and is updated using the State Estimator function $SE(b, a, o')$:
\[
b'(s') = SE(b, a, o') = Pr(s'|b, a, o') = \eta \Omega(a, s', o') \sum_{s \in S} b(s) T(s, a, s')
\]
where $\eta$ is the normalization factor, defined as:
\[
\eta = \frac{1}{Pr(o'|b, a)}
\]
and
\[
Pr(o'|b, a) = \sum_{s \in S} \left[ \Omega(a, s', o') \sum_{s \in S} b(s) T(s, a, s') \right]
\]
that is probability of observing $o'$ after performing action $a$ in the belief $b$. 

The reward model can also be defined on the beliefs:
\[
R(b, a) = \sum_{s \in S} b(s) R(s, a)
\]
Note, an important property of the belief state is that it is a sufficient statistics. In words, the belief at time $t$, i.e., $b_t$, summarizes the initial belief $b_0$, as well as all the actions taken and all observation received [Kaelbling et al., 1998]. Formally, we have:
\[
b_t(s) = Pr(s|b_0, a_0, o_0, \ldots, a_{t-1}, o_{t-1}).
\]
The POMDP policy selects an action $a$ for a belief state $b$, i.e., $a = \pi(b)$. In the POMDP framework the objective is to find an optimal policy $\pi^*$, where for any belief $b$,
\( \pi^* \) specifies an action \( a = \pi^*(b) \) that maximizes the expected discount of future rewards starting from belief \( b_0 \):

\[
V^\pi(b) = E_{b_t \sim SE} \left[ \gamma^0 R(b_0, \pi(b_0)) + \gamma^1 R(b_1, \pi(b_1)) + \ldots | \pi, b_0 = b \right]
\]

\[
= E_{b_t \sim SE} \left[ \sum_{t=0}^{\infty} \gamma^t R(b_t, \pi(b_t)) | \pi, b_0 = b \right]
\]

Similar to MDPs, the value function of a policy can also be recursively defined as:

\[
V^\pi(b) = E_{b_t \sim SE} \left[ \gamma^0 R(b_0, \pi(b_0)) + \gamma^1 R(b_1, \pi(b_1)) + \ldots | \pi, b_0 = b \right]
\]

\[
= E_{b_t \sim SE} \left[ \sum_{t=0}^{\infty} \gamma^t R(b_t, \pi(b_t)) | \pi, b_0 = b \right]
\]

\[
= R(b, \pi(b)) + \gamma E_{b_t \sim SE} \left[ \sum_{t=0}^{\infty} \gamma^t R(b_t, \pi(b_t)) | \pi, b_0 = b \right]
\]

The last equation is Bellman equation for POMDPs, defined as:

\[
V^\pi(b) = \left[ R(b, \pi(b)) + \gamma \sum_{o' \in O} Pr(o'|b, \pi(b)) V^\pi(b') \right] \tag{3.5}
\]

Then, we have the optimal policy \( \pi^* \) as:

\[
\pi^*(b) = \arg \max_{\pi} V^\pi(b)
\]

And the optimal belief-value model \( V^* \) can be found by:

\[
V^*(b) = \max_{\pi} V^\pi(b)
\]

\[
= \max_{\pi} \left[ R(b, \pi(b)) + \gamma \sum_{o' \in O} Pr(o'|b, \pi(b)) V^\pi(b') \right]
\]

We can also define Bellman value function as a function of beliefs and actions, \( Q^\pi(b, a) \), which estimates the expected return of taking action \( a \) in a given belief \( b \) and policy \( \pi \):

\[
Q^\pi(b, a) = R(b, a) + \gamma \sum_{o' \in O} Pr(o'|a, b) V^\pi(b')
\]
where \( b' = SE(b, a, o') \), is calculated from Equation (3.3).

Notice that we can see a POMDP as a MDP, if the POMDP includes a deterministic observation model and a deterministic initial belief. This can be seen in Equation (3.3), by starting with a deterministic initial belief, the next belief will be deterministic as the observation model is deterministic. This means that such a POMDP knows its current state with 100% probability similar to MDPs.

### 3.1.3 Reinforcement learning

In Section 3.1, we introduced model-free RL, in short RL, which is performed when the environment model is not known. An algorithm known as Q-learning [Watkins and Dayan, 1992] can be used for RL. These values estimate the expected return of taking action \( a \) in state \( s \) and following thereafter, as expressed in Equation (3.2). The process of policy learning in the Q-learning algorithm can be seen in the matrix of Table 3.1, taken from Schatzmann et al. [2006]. The Q-values, are initialized with an arbitrary value for every pair \((s, a)\). The Q-values are iteratively updated to become better estimates of the expected return of the state-action pairs. While the agent is interacting with the environment the Q-values are updated using:

\[
Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left( R(s, a) + \gamma \max_{a'}Q(s', a') \right)
\]

where \( \alpha \) represents a learning rate parameter that decays from 1 to 0. When the Q-values for each state action pair is estimated, the optimal policy for each state selects the action with the highest expected value, i.e., the bolded values in Table 3.1.

In this thesis, our focus is on learning the dialogue MDP/POMDP model components and then solve the dialogue MDP/POMDP using the available planning algorithms. As such, we study the planning algorithms for solving MDPs/POMDPs in the following section.

<table>
<thead>
<tr>
<th></th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
<th>( s_4 )</th>
<th>( s_5 )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>4.23</td>
<td>5.67</td>
<td>2.34</td>
<td>0.67</td>
<td>9.24</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>1.56</td>
<td>9.45</td>
<td>8.82</td>
<td>5.81</td>
<td>2.36</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( a_3 )</td>
<td>4.77</td>
<td>3.39</td>
<td>2.01</td>
<td>7.58</td>
<td>3.93</td>
<td>( \ldots )</td>
</tr>
</tbody>
</table>

**Table 3.1:** The process of policy learning in the Q-learning algorithm [Schatzmann et al., 2006].
3.1.4 Solving MDPs/POMDPs

Solving MDPs/POMDPs can be performed when the model components of the MDP or POMDP are defined/learned in advance. That is, solving the underlying MDP/POMDP for a near optimal policy. This is done by applying various model-based algorithms which work using dynamic programming [Bellman, 1957a]. Such algorithms fall into two categories of policy iteration and value iteration [Sutton and Barto, 1998]. In the rest of this section, we describe the policy iteration and value iteration for the MDP framework respectively in Section 3.1.4.1 and in Section 3.1.4.2. Then in Section 3.1.4.3, we introduce the value iteration for the POMDP framework. Since the value iteration algorithm for POMDPs is intractable, we study an approximated value iteration algorithm for the POMDP framework, known as point-based value iteration (PBVI) in Section 3.1.4.4.

3.1.4.1 Policy iteration for MDPs

Policy iteration methods have a general way of solving the value function in MDPs. They find the optimal value function by iterating on two phases known as policy evaluation and policy improvement shown in Algorithm 2. In Line 3, a random policy is selected, i.e., the policy \(\pi_t\) is randomly initialized at \(t = 0\). Then a random subsequent value of the policy is selected, i.e., the value \(V_k\) is randomly chosen when \(k = 0\). The algorithm then iterates on the two steps of policy evaluation and policy improvement.

In the policy evaluation step, i.e., Line 7, the algorithm calculates the value of policy \(\pi_{t+1}\). This is done efficiently by calculating the value of \(V_{k+1}\) using the value function \(V_k\) of previous policy \(\pi_t\), and then repeating this calculation until it finds a converged value for \(V_k\). This is formally done as follows:

\[
\forall s \in S : V_{k+1}(s) \leftarrow R(s, \pi_t(s)) + \gamma \sum_{s' \in S} T(s, \pi_t(s), s') V_k(s')
\]

The algorithm iterates until for all states \(s\) the state values stabilize. That is, we have: \(|V_k(s) - V_{k-1}(s)| < \epsilon\), where \(\epsilon\) is a predefined threshold for error.

Then, in the policy improvement step, i.e., Line 10, the greedy policy \(\pi_{t+1}\) is chosen. Formally, given the value function \(V_k\), we have:

\[
\forall s \in S : \pi_{t+1}(s) \leftarrow \arg \max_{a \in A} \left[ R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_k(s') \right]
\]

The process of policy evaluation and policy improvement continues until \(\pi_t = \pi_{t+1}\). Then, policy \(\pi_t\) is the optimal policy, i.e., \(\pi_t = \pi^*\).
Algorithm 2: The policy iteration algorithm for MDPs.

**Input:** An MDP model \( \langle S, A, T, R \rangle \);  
**Output:** A (near) optimal policy \( \pi^* \);

```plaintext
/* Initialization */
1 \( t \leftarrow 0; \)
2 \( k \leftarrow 0; \)
3 \( \forall s \in S: \) Initialize \( \pi_{t}(s) \) with an arbitrary action;
4 \( \forall s \in S: \) Initialize \( V_{k}(s) \) with an arbitrary value;
5 repeat
  /* Policy evaluation */
  6 repeat
    7 \( \forall s \in S: V_{k+1}(s) \leftarrow R(s, \pi_{t}(s)) + \gamma \sum_{s' \in S} T(s, \pi_{t}(s), s') V_{k}(s'); \)
    8 \( k \leftarrow k + 1; \)
  9 until \( \forall s \in S: |V_{k}(s) - V_{k-1}(s)| < \epsilon; \)
  /* Policy improvement */
  10 repeat
    11 \( t \leftarrow t + 1; \)
  12 until \( \pi_{t} = \pi_{t-1}; \)
13 \( \pi^* = \pi_{t}; \)
```

The significant drawback of the policy iteration algorithms is that for each improved policy \( \pi_{t} \), a complete policy evaluation is done (Line 7 and Line 8). Generally, value iteration algorithm is used to handle this drawback. We study value iteration algorithms for both MDPs and POMDPs in the following sections.

### 3.1.4.2 Value iteration for MDPs

Value iteration methods overlap the evaluation and improvement steps introduced in the previous section. Algorithm 3 demonstrates the value iteration method in MDPs. It consists of a backup operation as:

\[
\forall s \in S: V_{k+1}(s) \leftarrow \max_{a \in A} \left[ R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_{k}(s') \right]
\]

This operation continues in Line 4 and Line 5 until for all states \( s \), state values stabilize. That is, we have: \( |V_{k}(s) - V_{k-1}(s)| < \epsilon \). Then, the optimal policy is the greedy policy with regard to the value function shown in Line 4.
Algorithm 3: The value iteration algorithm for MDPs.

**Input:** An MDP model \(\langle S, A, T, R \rangle\);

**Output:** A (near) optimal policy \(\pi^*\);

1. \(k \leftarrow 0;\)
2. \(\forall s \in S: \text{Initialize } V_k(s) \text{ with an arbitrary value};\)
3. repeat
   4. \(\forall s \in S: V_{k+1}(s) \leftarrow \max_{a \in A} \left[ R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_k(s') \right];\)
   5. \(k \leftarrow k + 1;\)
4. until \(\forall s \in S: |V_k(s) - V_{k-1}(s)| < \epsilon;\)
5. \(\forall s \in S: \pi^*(s) \leftarrow \arg \max_{a \in A} \left[ R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_k(s') \right];\)

3.1.4.3 Value iteration for POMDPs

Solving POMDPs is more challenging than solving MDPs. To find the solution of a MDP, an algorithm such as value iteration needs to find the optimal policy for \(|S|\) discrete states. However, finding the solution of POMDPs is more challenging, since the algorithm, such as value iteration, needs to find the solution for \(|S| - 1\) dimensional continuous space. This problem is called curse of dimensionality in POMDPs [Kaelbling et al., 1998]. Then, the POMDP solution is found as a breadth first search in \(t\)-steps, for the beliefs that have been created in the \(t\)-steps. This is called \(t\)-step planning. Notice that the number of created beliefs increases exponentially with respect to the planning time \(t\). This problem is called curse of history in POMDPs [Kaelbling et al., 1998; Pineau, 2004].

Planning is performed in POMDPs as a breadth first search in trees for a finite \(t\), and consequently finite \(t\)-step conditional plans. A \(t\)-step conditional plan describes a policy with a horizon of \(t\)-step further [Williams, 2006]. It can be represented as a tree that includes a specified root action \(a_t\). Figure 3.2 shows a 3-step conditional plan in which the root is indexed with time step \(t\) (\(t = 3\)) and the leaves are indexed with time step 1. The edges are indexed with observations that lead to a node at \(t - 1\) level, representing a \(t - 1\)-step conditional plan.

Each \(t\)-step conditional plan has a specific value \(V_t(s)\) for unobserved state \(s\) which is calculated as:

\[
V_t(s) = \begin{cases} 
0 & \text{if } t = 0; \\
R(s, a_t) + \gamma \sum_{s' \in S} T(s, a_t, s') \sum_{o' \in O} \Omega(a_t, s', o') V_{t-1}'(s') & \text{otherwise};
\end{cases}
\]

where \(a_t\) is the specified action for the \(t\)-step conditional plan. Moreover, \(V_{t-1}'(s')\) is
the value of $t-1$-step conditional plan (in level $t-1$) which is the child index $o'$ of $t$ conditional plan (with root node $a_t$).

Since in POMDPs the state is unobserved and a belief over possible states are maintained then the value of $t$-step conditional plan is calculated in runtime using the current belief $b$. More specifically, the value of $t$-step conditional plan for belief $b$, denoted by $V_t(b)$, is an expectation over states:

$$V_t(b) = \sum_{s \in S} b(s)V_t(s)$$

In POMDPs, given a set of $t$-step conditional plans, the agent’s task is to find the conditional plan that maximizes the belief’s value. Formally, given a set of $t$-step conditional plans denoted by $N_t$, in which the plans’ indices are denoted by $n$, the best $t$-step conditional plan is the one that maximizes the belief’s value:

$$V_t^*(b) = \max_{n \in N_t} \sum_{s \in S} b(s)V_t^n(s) \quad (3.6)$$

where $V_t^n$ is the $n$th $t$-step conditional plan.

And, the optimal policy for belief $b$ is calculated as:

$$\pi^*(b) = a_t^n$$

where $n = \arg \max_{n \in N_t} \sum_{s \in S} b(s)V_t^n(s)$.

The value of each $t$-step conditional plan, $V_t(b)$, is a hyperplane in belief state, since it is an expectation over states. Moreover, the optimal policy takes the max over many hyperplanes, this causes the value function, Equation (3.6), to be piece-wise-linear and convex. The optimal value function is then formed of regions where one hyperplane (one conditional plan) is optimal [Sondik, 1971; Smallwood and Sondik, 1973].
After this introduction of planning for POMDPs, now we can go through value iteration in POMDPs. Algorithm 4, adapted from [Williams, 2006], describes value iteration for POMDPs [Monahan, 1982; Kaelbling et al., 1998]. Value iteration proceeds by finding the subset of possible $t$-step conditional plans which contribute to the optimal $t$-step policy. These conditional plans are called *useful*, and only useful $t$-step plans are considered when finding the $(t + 1)$-step optimal policy. In this algorithm, the input is a POMDP model and the planning time $\text{max}T$, and the output is the set of $\text{max}T$-step conditional plans, denoted by $V^n_{\text{max}T}$, and their subsequent actions, denoted by $a^n_{\text{max}T}$.

Each iteration of the algorithm contains two steps of generation and pruning. In the generation steps, Line 4 to Line 11, the possibly useful $t$-step conditional plans are generated by enumerating all actions followed by all possible useful combinations of $(t - 1)$-step conditional plans. This is done in Line 8:

$$
\begin{align*}
    v^{a,k} & \leftarrow R(s, a) + \gamma \sum_{s' \in S} \sum_{o' \in O} T(s, a, s') \Omega(a, s', o') V_{t-1}^k(o') \\
& \text{where } k(o') \text{ refers to element } o' \text{ of the vector } k = (V_{t-1}^{n_1}, \ldots, V_{t-1}^{n_{|O|}}).
\end{align*}
$$

Then, pruning is done in Line 12 to Line 25. In the pruning step, the conditional plans that are not used in the optimal $t$-step policy are removed, which remains the set of useful $t$-step conditional plans. In particular, in Line 16, if there is a belief where $v^{a,k}$ makes the optimal policy, then the $n$th index of $t$-step conditional plan is set to $v^{a,k}$, i.e., $V^n_t(s) = v^{a,k}$.

Notice that value iteration for POMDPs is exponential to the number of observations [Cassandra et al., 1995]. In fact, it has been proved that finding the optimal policy of a POMDP is a PSPACE-complete problem [Papadimitriou and Tsitsiklis, 1987; Madani et al., 1999]. Even finding a near optimal policy, i.e., a policy with a bounded value loss compared to the optimal one is NP-hard for a POMDP [Lusena et al., 2001].

As introduced in the beginning of this section, the main challenge for planning in POMDPs is because of curse of dimensionality and curse of history. So, numerous approximate algorithms for planning in POMDPs have been proposed in the past. For instance, Smallwood and Sondik [1973] developed a variation of value iteration algorithm for POMDPs. Other approaches include point-based algorithms [Pineau et al., 2003; Pineau, 2004; Smith and Simmons, 2004; Spaan and Spaan, 2004; Paquet et al., 2005], heuristic-based method of Hauskrecht [2000], structure-based algorithms [Bonet and Geffner, 2003; Dai and Goldsmith, 2007; Dibangoye et al., 2009], compression-based algorithms [Lee and Seung, 2001; Roy et al., 2005; Poupart and Boutilier, 2002; Li et al., 2007], and forward search algorithms [Paquet, 2006; Ross et al., 2008]. In this context, the point-based value iteration algorithms [Pineau et al., 2003] perform the planning for a fixed set of belief points. In the following section, we study the PBVI algorithm.
Algorithm 4: The value iteration algorithm in POMDPs adapted from Williams [2006].

**Input:** A POMDP model $\langle S, A, T, \gamma, R, O, \Omega, b_0 \rangle$ and $\text{max}T$ for planning horizon;

**Output:** The conditional plan $V^{\text{max}T}$ and its subsequent action $a^{\text{max}T}$:

1. $\forall s \in S$: Initialize $V_0(s)$ with 0;
2. $N \leftarrow 1$;
   /* $N$ is the number of $t-1$ step conditional plans */
3. **for** $t \leftarrow 1$ to $\text{max}T$ **do**
   /* Generate $\{v^{a,k}\}$, the set of possibly useful conditional plans */
4.   $K \leftarrow \{V^n_{t-1}: 1 \leq n \leq N\}^{(O)}$;
   /* $K$ now contains $N^{(O)}$ elements, where each element $k$ is a vector $k = (V^{x_1}_{t-1}, \ldots, V^{x_{(O)}}_{t-1})$. This growth is the source of the computational complexity */
5.   **foreach** $a \in A$ **do**
6.     **foreach** $k \in K$ **do**
7.         **foreach** $s \in S$ **do**
8.             /* Notation $k(o')$ refers to element $o'$ of vector $k$. */
9.             $v^{a,k}(s) \leftarrow R(s,a) + \gamma \sum_{s' \in S} \sum_{o' \in O} T(s,a,s')\Omega(a,s',o')V^{k(o')}_{t-1}(s')$;
10.            end
11.       end
12.   end
13.   /* Prune $\{v^{a,k}\}$ to yield $\{V^n\}$, set of actually useful CPs */
14.   /* $n$ is the number of $t$-step conditional plans */
15.   $n \leftarrow 0$;
16. **foreach** $a \in A$ **do**
17.     **foreach** $k \in K$ **do**
18.         /* If the value of plan $v^{a,k}$ is optimal in any belief, it is useful and will be kept. */
19.         if $\exists b: v^{a,k}(b) = \max_{a,k} v^{a,k}(b)$ then
20.             $n \leftarrow n + 1$;
21.             $a^n_s \leftarrow a$;
22.             **foreach** $s \in S$ **do**
23.                 $V^n_s(s) \leftarrow v^{a,k}(s)$;
24.             end
25.         end
26. end
27. $N \leftarrow n$;

as described in [Williams, 2006].
3.1.4.4 Point-based value iteration for POMDPs

Value iteration for POMDPs is computationally complex, because it tries to find an optimal policy for all belief points in the belief space. As such, not all of the generated conditional plans (in the generation step of value iteration) can be processed in the pruning step. In fact, in the pruning step there is a search for a belief in continuously-valued space of beliefs [Williams, 2006]. On the other hand, the PBVI algorithm [Pineau et al., 2003] works by searching optimal conditional plans only at a finite set of \( N \) discrete belief points \( \{b_1, \ldots, b_N\} \). That is, each unpruned conditional plan \( V^n_t(s) \) is exact only at belief \( b_n \), and consequently PBVI algorithms are approximate planning algorithms for POMDPs\(^1\).

Algorithm 5 adapted from [Williams, 2006] describes the PBVI algorithm. The input and output of the algorithm is similar to the value iteration algorithm for POMDPs. Here, the input adds a set of \( N \) random discrete belief points (besides the POMDP model and the planning time \( maxT \) which is used also in value iteration for POMDPs). And, the output is the set of \( maxT \)-step conditional plans, denoted by \( V_{maxT}^n \), and their subsequent actions, denoted by \( a_{maxT}^n \).

Similar to value iteration for POMDPs, the PBVI algorithm consists of two steps of generation and pruning. In Line 7 to Line 17, the possibly useful \( t \)-step conditional plans are generated using the \( N \) given belief points to the algorithm. First, for each given belief point, the next belief is formed for all possible action observation pairs; denoted by \( b_n^{a,o'} \) in Line 10. Then, for each updated belief, \( b_n^{a,o'} \), the index of the best \( t-1 \)-step conditional plan is stored; denoted by \( m(o') \) in Line 11. That is, the \( t-1 \)-step conditional plan that brings the highest value for the updated belief, which is calculated as:

\[
m(o') \leftarrow \arg \max_{n_i} \sum_{s' \in S} b_n^{a,o'} (s') V_{t-1}^{m_n}(s')
\]

The final task in the generation step of PBVI is generating a set of possible useful conditional plan for the current belief and action, denoted by \( v^{a,n} \) which is calculated for each state in Line 14 as:

\[
v^{a,n}(s) \leftarrow R(s, a) + \gamma \sum_{s' \in S} \sum_{o' \in O} T(s, a, s') \Omega(a, s', o') V_{t-1}^{m(o')}(s')
\]

where \( V_{t-1}^{m(o')}(s') \) is the best \( t-1 \)-step conditional plan for the updated belief \( b_n^{a,o'} \).

Finally, the pruning step is done in Line 18 to Line 23. In the pruning step, for each given belief point \( n \), the highest valued conditional plan is selected and the rest ones are

\(^1\)Note that here we assume that the PBVI is performed on a fixed set of random points similar to the PERSEUS algorithm, the point-based value iteration algorithm proposed by Spaan and Vlassis [2005].
Algorithm 5: Point-based value iteration algorithm for POMDPs adapted from Williams [2006].

Input: A POMDP model $\langle S, A, T, \gamma, R, O, \Omega, b_0 \rangle$, $maxT$ for planning horizon, and a set of $N$ random beliefs $B$;

Output: The conditional plan $V_{maxT}^n$ and its subsequent action $a_{maxT}^n$;

1 for $n \leftarrow 1$ to $N$ do
2    foreach $s \in S$ do
3        $V_0^n(s) \leftarrow 0$;
4    end
5 end
6 for $t \leftarrow 1$ to $T$ do
7    /* Generate $\{v^{a,k}\}$, the set of possibly useful conditional plans */
8    for $n \leftarrow 1$ to $N$ do
9        foreach $a \in A$ do
10           foreach $o' \in O$ do
11              $b_{n,a,o'} \leftarrow SE(b_n, a, o')$;
12              $m(o') \leftarrow \arg \max_{n_i} \sum_{s' \in S} b_{n,a,o'}(s') V_{t-1}^{m_i}(s')$;
13           end
14        end
15        foreach $s \in S$ do
16           $v^{a,n}(s) \leftarrow R(s, a) + \gamma \sum_{s' \in S} \sum_{o' \in O} T(s, a, s') \Omega(a, s', o') V_{t-1}^{m(o')}(s')$;
17        end
18    end
19    /* Prune $\{v^{a,n}\}$ to yield $\{V_t^n\}$, set of actually useful CPs */
20    for $n \leftarrow 1$ to $N$ do
21       $a_t^n \leftarrow \arg \max_{a} \sum_{s \in S} b_n(s) v^{a,n}(s)$;
22       foreach $s \in S$ do
23          $V_t^n(s) \leftarrow v^{a_t^n,n}(s)$;
24       end
25    end
26 end

pruned, in Line 19. This is done by finding the best action (the best $t$-step policy) from the generated conditional beliefs for the belief point $n$, i.e., $v^{a,n}$, which is calculated as:

$$a_t^n \leftarrow \arg \max_{a} \sum_{s \in S} b_n(s) v^{a,n}(s)$$

and its subsequent $t$-step conditional plan is stored as $V_t^n$ in Line 21.
In contrast to value iteration for POMDPs, the number of conditional plans are fixed in all iterations in the PBVI approach (which is equal to the number of the given belief points, $N$). This is because of the fact that each conditional plan is optimal at one of the belief points. Notice that although the set of found conditional plans are guaranteed to be optimal only at the finite set of given belief points, the hope is that they are optimal (or near optimal) for nearby belief points. Then, similar to value iteration the conditional plan for an arbitrary belief $b$ at run time is calculated using $\max_n b(s)V^*_n(s)$.

### 3.2 Spoken dialogue management

The spoken dialogue system (SDS) of an intelligent machine is the system that is responsible for the interaction between machine and human users. Figure 3.3, adapted from Williams [2006], shows the architecture of an SDS. At the high level, an SDS consists of three modules: the input, the output, and the control. The input includes the automatic speech recognition (ASR) and natural language understanding (NLU) components. The output includes natural language generator (NLG) and text-to-speech (TTS) components. Finally, the control module is the core part of an SDS and consists of the dialogue model and the dialogue manager (DM). The control module is also called the dialogue agent in this thesis.

The SDS modules work as follows. First, the ASR module receives the user utterance, i.e., a sequence of words in the form of speech signals, and makes a N-Best list containing all user utterance hypotheses. Next, NLU receives the noisy words from the ASR output, generates the possible intentions that the user could have in mind, and sends them to the control module. The control module receives the generated user intentions, possibly with a confidence score, as an observation $O$. The confidence score can show for instance the reliability of possible user intentions since the output generated by ASR and NLU can cause uncertainty in the machine. That is, the ASR output includes errors and the NLU output can be ambiguous, both cause uncertainty in SDS. The observation $O$ can be used in a dialogue model to update and enhance the model.

Notice that the dialogue model and the dialogue manager interact with each other. In particular, the dialogue model provides the dialogue manager with the observation $O$ and the updated model. Based on such information, the dialogue manager is responsible for making a decision. In fact, the DM updates its strategy based on the received updated model, and refers to its strategy for producing an action $A$, which is an input for NLG. The task of NLG is to produce a text describing the action $A$, and to pass the text to the TTS component. Finally, the TTS produces the spoken utterance of the text, and announces it for the user.
Chapter 3. Sequential decision making in spoken dialogue management

Figure 3.3: The architecture of a spoken dialogue system, adapted from Williams [2006].

Note also that the dialogue control part is the core part of an SDS, and is responsible for holding an efficient and natural communication with the user. To do so, the environment dynamics are approximated in the dialogue model component over time. In fact, the dialogue model aims to provide the dialogue manager with better approximates of the environment dynamics. More importantly, the dialogue manager is required to learn a strategy based on the updated model and to make a decision that satisfies the user intention during the dialogue. But, this is a difficult task primarily because of the noisy ASR output, the NLU difficulties, and also the user intention change during the dialogue. Thus, model learning and decision making is a significant task in SDS. In this context, the spoken dialogue community modeled the dialogue control of an SDS in the MDP/POMDP framework to automatically learn the dialogue strategy, i.e., the dialogue MDP/POMDP policy.

3.2.1 MDP-based dialogue policy learning

In the previous section, we studied that the control module of an SDS is responsible for dialogue modeling and management. The control module of a spoken dialogue system, i.e., the dialogue agent, has been formulated in the MDP framework so that the dialogue MDP agent learns the dialogue policy [Pieraccini et al., 1997; Levin and Pieraccini, 1997]. In this context, the MDP policy learning can be done either via model-free RL,
or model-based RL. The model-free RL, in short RL, introduced in Section 3.1.3, can be done using techniques such as Q-learning. The model-based dialogue policy learning is basically solving the dialogue MDP/POMDP model using algorithms such as value iteration, introduced in Section 3.1.4.

In the model-based dialogue policy learning, the dialogue MDP model components can be given either by the domain experts manually, or learned from dialogues. In particular, the supervised learning approach can be used after annotating a dialogue set to learn user models. For example, a user model can encode the probability of changing the user intention in each turn, given an executed machine’s action. We study the user models further in Section 3.2.3. Then, the dialogue MDP policy is learned using algorithms such as the value iteration algorithm, introduced in Section 3.1.4.2.

On the other hand, in the model-free RL which is also called simulation-based RL [Rieser and Lemon, 2011], the dialogue set is annotated and used for learning a simulated environment. Figure 3.4, taken from Rieser and Lemon [2011], shows a simulated environment. The dialogue set is first annotated, and then used to learn the user model using supervised learning techniques. Moreover, the simulated environment requires an error model. The error model encodes the probability of occurring errors, for example by the ASR machine. The error model can be learned also from the dialogue set. Then, model-free MDP policy learning techniques such as Q-learning (Section 3.1.3) is applied to learn the dialogue MDP policy through interaction with the simulated user. For a comprehensive survey of recent advances in MDP-based dialogue strategy learning (particularly simulation-based learning) the interested readers are referred to Frampton and Lemon [2009].

In contrast to MDPs, POMDPs are more general stochastic models that do not assume the environment’s states fully observable, as introduced in Section 3.1. Instead, observations in POMDPs provide only partial information to the machine, and consequently, POMDPs maintain a belief over the states. As a result, the dialogue POMDP policy performance is substantially higher than that of the dialogue MDP policies, particularly in the noisy environments [Gašić et al., 2008; Thomson and Young, 2010].

In this context, the POMDP-based dialogue strategy learning is mostly model-based [Kim et al., 2011]. This is mainly because reinforcement learning in POMDPs is a hard problem, and it is still being actively studied [Wierstra and Wiering, 2004; Ross et al., 2008, 2011]. In the next section, we present the related research on dialogue POMDP policy learning.
Figure 3.6: Simulation-based RL: Learning a stochastic simulated dialogue environment from data [Rieser and Lemon, 2011].

3.2.2 POMDP-based dialogue policy learning

The pioneer research for application of POMDPs in SDSs has been performed by Roy et al. [2000]. The authors defined a dialogue POMDP for spoken dialogue system of a robot by considering possible user intentions as the POMDP states. More specifically, their POMDP contained 13 states with mixture of 6 user intentions and several user actions. In addition, the POMDP actions included 10 clarifying questions as well as performance actions such as going to a different room, and presenting information to user.

For the choice of observations, the authors defined 15 *keywords* and an observation for the *nonsense* words. Moreover, the choice of the reward model has been hand-tuned. In fact, their defined reward model returned -1 for each dialogue turn, that is for each clarification question regardless of the state of POMDP.

Then, Zhang et al. [2001b] proposed a dialogue POMDP in the tourist guide domain. Their POMDP included 30 states with two factors, one factor with 6 possible user intentions. The other factor encoded 5 values indicating the channel error such as *normal*, and *noisy*. For the choice of the POMDP actions, the authors defined 18 actions such as *Asking user’s intention* and *Confirming user’s intention*.

Also, for the choice of the POMDP observations, Zhang et al. [2001b] defined 25 ob-
servations for the statement of user’s intention, for instance yes, no, and no response. Moreover, for the reward model, they used a small negative reward for asking the user’s intention, a large positive reward for presenting the right information for the user’s intention, and a large negative reward, otherwise. Finally, they used approximated methods to find their defined dialogue POMDP solution and concluded that the POMDP approximate solution outperforms an MDP baseline.

Williams and Young [2007] also formulated the control module of spoken dialogue systems in the POMDP framework. They factorized the machine’s state to three components:

\[ s = (g, u, d) \]

where \( g \) is the user goal, which is similar to user intention, \( u \) is the user action, i.e., the user utterance. In addition, \( d \) is the dialogue history, which indicates, for instance, what the user has said so far, or the user’s view of what has been grounded in the conversation so far [Clark and Brennan, 1991; Traum, 1994]. For a travel domain, the user goal could be any possible (origin, destination) pair allowed in the domain for instance (London, Edinburgh). Moreover, the user utterances could be similar to from London to Edinburgh. Finally, the machine’s action could be such as WHICH ORIGIN, and WHICH DESTINATION.

Williams and Young [2007] assumed that the user goal at each time step depends on the user goal and the machine’s action in the previous time step:

\[ Pr(g'|g, a) \]

Moreover, they assumed that the user’s action depends on the user goal and machine’s action in the previous time step:

\[ Pr(u'|g', a) \]

Furthermore, the authors assumed that the current dialogue history depends on the user goal and action, as well as the dialogue history and the machine’s action in the previous time step:

\[ Pr(d'|u', g', d, a) \]

Then, the state transition becomes:

\[
Pr(s'|s, a) = Pr(g'|g, a) \cdot Pr(u'|g', a) \cdot Pr(d'|u', g', d, a) \tag{3.7}
\]

For the observation model, Williams and Young [2007] used the noisy recognized user’s utterance \( \hat{u} \) together with confidence score \( c \):

\[ o = (u', c) \]
Moreover, they assumed that the machine’s observation is based on the user’s utterance and the confidence score $c$:

$$p(o'|s', a) = p(\tilde{u}', c'|u)$$

In addition, Williams and Young [2007] used a hand-coded reward model, for instance, large negative rewards for Asking a Non-relevant Question, small negative reward for Confirmation actions, and positive reward for Ending the dialogue successfully. In this way, the learned dialogue POMDP policies try to minimize the number of turns and at the same time to finish a successful dialogue.

Doshi and Roy [2007, 2008] proposed a dialogue POMDP for a spoken dialogue system of a robot. Similar to Roy et al. [2000], the authors considered the user’s intention as POMDP states, for instance the user’s intention for coffee machine area, or main elevator. In addition, they defined machine actions such as Where would you like to go, and What would you like. Furthermore, the observations are the user utterances, for instance I would like coffee. In this work, the transition model encodes the probability of keywords given the machine’s actions. For instance, given the machine’s action Where do you want to go, there is a high probability that the machine receives coffee, or coffee machine. Doshi and Roy [2008] used Dirichlet priors for uncertainty in the transition and observation models. In particular, for observation model they used Dirichlet counts and used an HMM to find the underlying states using EM algorithm.

Note that there are numerous other related works on dialogue POMDPs. For instance, [Doshi and Roy, 2008; Doshi-Velez et al., 2012] used active learning for learning dialogue POMDPs. [Thomson, 2009; Thomson and Young, 2010; Png and Pineau, 2011; Atrash and Pineau, 2010] used Bayesian techniques for learning dialogue POMDP model components. In this context, Atrash and Pineau [2010] introduced a Bayesian method of learning an observation model for POMDPs which is explained further in Section 4.4. Moreover, Png and Pineau [2011] proposed an online Bayesian approach for updating the observation model of dialogue POMDPs which is also described further in Section 4.4.

As mentioned, the learned dialogue POMDP model components affect the optimized policy of the dialogue POMDP. In particular, the transition model of a dialogue POMDP usually includes the user model which needs to be learned from the dialogue set. Kim et al. [2008] described different user model techniques that have been used in dialogue POMDPs. These models are described in the following section.
3.2.3 User modeling in dialogue POMDPs

In this section, we described the four user modeling techniques that have been used in dialogue POMDPs [Kim et al., 2011]. These models include \textit{n-grams} (particularly the \textit{bi-grams} and \textit{tri-grams}) [Eckert et al., 1997], the Levin model [Levin and Pieraccini, 1997], the Pietquin model [Pietquin, 2004], and the \textit{HMM} user model [Cuayáhuitl et al., 2005].

The bi-gram model learns the probability that the user performs action \( u \), given the machine executes action \( a \):

\[
Pr(u|a)
\]

In tri-grams, the machine actions in two previous time-steps are considered. That is, the tri-gram model learns:

\[
Pr(u|a_n, a_{n-1})
\]

The \( n \)-grams are simple models to develop, however, their drawback is that the number of parameters can be large.

Thus, the Levin model reduces the number of parameters in the bi-grams by considering the \textit{type} of the machine’s action and learning the user actions for each type. These types include: \textit{greeting}, \textit{constraining}, and \textit{relaxing} actions. The greeting action could be for instance \textit{How can I help you}? The constraining actions are used to constraint a slot, for instance \textit{From which city are you leaving}? The relaxing actions are used for relaxing a constraint from a slot, for instance \textit{do you have other dates for leaving}?

For the greeting action, the model learns:

\[
Pr(n)
\]

where \( n \) shows the number of slots for which the user provides info \((n = 0, 1, \ldots)\). Also, the model learns the distribution on each slot \( k \):

\[
Pr(k)
\]

where \( k \) is the slot number \((k = 1, 2, \ldots)\).

For the constraining actions, the model learns two probability models. One is the probability that the user provides value for \( n \) other slots while asked for slot \( k \):

\[
Pr(n|k)
\]

The other is the probability that the user provides value for slot \( k' \) when it is ask for slot \( k \):

\[
Pr(k'|k)
\]
For the relaxing actions, the user either accepts the relaxation of the constraint or rejects it. So for each slot, the model learns:

\[ Pr(\text{yes}|k) = 1 - Pr(\text{no}|k) \]

In the Levin model, however, the user goal is not considered in the user model. Then, the Pietquin model learns the probabilities conditioned on the user goal:

\[ Pr(u|a,g) \]

where \( u \) is the user action (utterance), \( g \) the user goal, and \( a \) the machine’s action. In this model the user goal is represented as a table of slot-value pairs. Since this can be a large table, an alternative approach can be considered. That is, for each part of the user goal, which is each slot, it is only maintained whether or not the user has provided information for that slot. So, for a dialogue model with 4 slots, there exist only \( 2^4 = 16 \) user goals. Note that in this way of user modeling the goal consistency is not maintained in the same way as the original Pietquin model.

In the HMM user modeling, first the probability of executing the machine’s actions is learned based on the dialogue state:

\[ Pr(a|d) \]

where \( d \) is for the dialogue state. Then, in the input HMM model, called IHMM, the model is enhanced by considering also the user actions besides the dialogue state:

\[ Pr(a|d, u) \]

Finally, in the input output HMM, IOHMM, the user action model is learned based on the dialogue state and the machine’s action:

\[ Pr(u|d, a) \]

Note that in the above mentioned works, the models are either assumed or have been learned from an annotated dialogue set. In the following chapter, we propose methods for learning the dialogue POMDP model components particularly the transition and observation models using unannotated dialogues and thus unsupervised learning techniques. Similar to Roy et al. [2000] and Doshi and Roy [2008], we use the user intentions as POMDP states in this thesis. However, here we are interested in learning the dialogue intentions from the dialogue set, rather than manually assigning them, and modeling the transition and observation models also based on unannotated dialogues.
Chapter 4

Dialogue POMDP model learning

4.1 Introduction

In this chapter, we propose methods for learning the model components of intention-based dialogue POMDPs from unannotated and noisy dialogues. As stated in Chapter 1, in intention-based dialogue domains, the dialogue state is the user intention, where the users can mention their intentions in different ways. In particular, we automatically learn the dialogue states by learning the user intentions from dialogues available in a domain of interest. We then learn a maximum likelihood transition model from the learned states. Furthermore, we propose two learned observation sets, and their subsequent observation models. The reward model however is learned in the next chapter where we present the IRL background and our proposed POMDP-IRL algorithms.

Note that we do not learn the discount factor since it is a number between 0 and 1 which is usually given. From the value function, shown in Equation (3.5), we can see that if the discount factor is equal to 0, then the MDP/POMDP optimizes only immediate rewards, whereas if it is equal to 1, then the MDP/POMDP is in favor of future rewards [Sutton and Barto, 1998]. In SDS, for instance Kim et al. [2011] set the discount factor to 0.95 for all their experiments. We also hand tuned the discount factor to 0.90 for all our experiments. We set the initial belief state to the uniform distribution in all our experiments.

In the rest of this chapter, in Section 4.2, we learn the dialogue POMDP states. In this section, we first describe an unsupervised topic modeling approach known as hidden topic Markov model (HTMM) [Gruber et al., 2007]; the method that we adapted for learning user intentions from dialogues, in Section 4.2.1. We then present an illustrative example, using SACTI-1 dialogues [Williams and Young, 2005], which shows the application of HTMM on dialogues for learning the user intentions, in Section 4.2.2. We introduce our maximum likelihood transition model using the learned intentions in
Section 4.3. Then, we propose two observation sets and their subsequent observation models, learned from dialogues, in Section 4.4. We then revisit through the illustrative example on SACTI-1 to apply the proposed methods for learning and training a dialogue POMDP (without the reward model) in Section 4.5. In this section, we also evaluate the HTMM method for learning dialogue intentions, in Section 4.5.1, followed by the evaluation of the learned dialogue POMDPs from SACTI-1 in Section 4.5.2. Finally, we conclude this chapter in Section 4.6.

4.2 Learning states as user intentions

Recall our Algorithm 1, presented in Chapter 1, that shows the high level procedure for dialogue POMDP model learning. The first step of the algorithm is to learn the states using an unsupervised learning method. As discussed earlier, the user intentions are used as the dialogue POMDP states. As such, in the first step we aim to capture the possible user intentions in a dialogue domain based on unannotated and noisy dialogues. Figure 4.1 represents dialogue states as they are learned based on an unsupervised learning (UL) method. Here, we use hidden topic Markov model (HTMM) [Gruber et al., 2007] to consider the Markovian property of states between $n$ and $n+1$ time steps. The HTMM method for intention learning from unannotated dialogues is as follows.

4.2.1 Hidden topic Markov model for dialogues

Hidden topic Markov model, in short HTMM [Gruber et al., 2007], is an unsupervised topic modeling technique that combines LDA (cf. Section 2.2) and HMM (cf. Section 2.3) to obtain the topics of documents. In Chinaei et al. [2009], we adapted

![Figure 4.1](image)

**Figure 4.1:** Hidden states are learned based on an unsupervised learning (UL) method that considers the Markovian property of states between $n$ and $n+1$ time steps. Hidden states are represented in the light circles.
HTMM for dialogues. A dialogue set $D$ consists of an arbitrary number of dialogues, $d$. Similarly, each dialogue $d$ consists of the recognized user utterances, $\tilde{u}$, i.e., the ASR recognition of the actual user utterance $u$. The recognized user utterance, $\tilde{u}$, is a bag of words, $\tilde{u} = [w_1, \ldots, w_n]$.

Figure 4.2 shows the HTMM model, which is similar to the LDA model shown in Figure 2.2. HTMM, however, applies the first-order Markov property to LDA, and is explained further in this section. Figure 4.2 shows that the dialogue $d$ in a dialogue set $D$ can be seen as a sequence of words $w_i$ which are observations for a hidden intentions $z$. Since hidden intentions are equivalent to user intentions, hereafter, hidden intentions are called user intentions. The vector $\beta$ is a global vector that ties all the dialogues in a dialogue set $D$, and retains the probability of words given user intentions, $Pr(w|z, \beta) = \beta_{wz}$. In particular, the vector $\beta$ is drawn based on multinomial distributions with a Dirichlet prior $\eta$. On the other hand, the vector $\theta$ is a local vector for each dialogue $d$, and retains the probability of intentions in a dialogue, $Pr(z|\theta) = \theta_{z}$. Moreover, the vector $\theta$ is drawn based on multinomial distributions with a Dirichlet prior $\alpha$.

The parameter $\psi_i$ is for adding the Markovian property in dialogues since successive utterances are more likely to include the same user intention. The assumption here is that a recognized utterance represents only one user intention, so all the words in the recognized utterance are observations for the same user intention. To formalize that, the HTMM algorithm assigns $\psi_i = 1$ for the first word of an utterance, and $\psi_i = 0$ for the rest. Then, when $\psi_i = 1$ (beginning of an utterance) a new intention is drawn, and when $\psi_i = 0$ (in the utterance), the intention of the $n$th word is identical to the intention of the previous one. Note that the parameter $\epsilon$ is used as a prior over $\psi$ which controls the probability of intention transition between utterances in dialogues, $Pr(z_i|z_{i-1}) = \epsilon$. Since each recognized utterance contains one user intention, we have $Pr(z_i|z_{i-1}) = 1$ for $z_i, z_{i-1}$ within one utterance.

Algorithm 6 is the generative algorithm for HTMM, adapted from Gruber et al. [2007]. This generative algorithm here is similar to the generative model of LDA introduced in Section 2.2. First, for all possible user intentions, the vector $\beta$ is drawn using the Dirichlet distribution with prior $\eta$, in Line 2. Then, for each dialogue, the vector $\theta$ is drawn using the Dirichlet prior $\alpha$. In Line 5, for each dialogue, the vector $\theta$ is initialized using the Dirichlet prior $\alpha$.

In HTMM, however, for each recognized utterance $i$ in dialogue $d$, the parameter $\psi$ is initialized based on a Bernoulli prior $\epsilon$ in Line 7 to Line 13. As mentioned above, the parameter $\psi$ basically adds the Markovian property to the model. It determines whether the user intention for the recognized utterance $i$ is the same as previous recognized utterance. The rest of the algorithm, Line 14 to Line 21, finds the user intentions.
Figure 4.2: The HTMM model adapted from Gruber et al. [2007], the shaded nodes are words ($w$) used to capture intentions ($z$).

If the parameter $\psi$ is equal to 0 the algorithm assumes that the user intention for utterance $i$ is equal to the one for utterance $i-1$, in Line 16, encoding thus the Markovian property. Otherwise, it draws the intention for utterance $i$ based on the vector $\theta$ in Line 18. Finally, a new word $w$ is generated based on the vector $\beta$, in Line 20.

HTMM uses Expectation Maximization (EM) and forward backward algorithm [Rabiner, 1990] (cf. Section 2.3), the standard method for approximating the parameters in HMMs. This is due to the fact that conditioned on $\theta$ and $\beta$, HTMM is a special case of HMMs. In HTMM, the latent variables are user intentions $z_i$ and $\psi_i$ which determines if the intention for the word $w_i$ is drawn from $w_{i-1}$, i.e., if $\psi_i = 0$; or a new intention will be generated, i.e., if $\psi_i = 1$.

1. In the expectation step, the $Q$ function from Equation (2.5) is instantiated. For each user intention $z$, we need to find the expected count of intention transitions to intention $z$.

$$E(C_{d,z}) = \sum_{j=1}^{[d]} Pr(z_{d,j} = z, \psi_{d,j} = 1|w_1, \ldots, w_{|d|})$$

where $d$ is a dialogue in the dialogue set, $D$. 
Algorithm 6: The HTMM generative model, adapted from Gruber et al. [2007].

Input: Set of dialogues $D$, $N$ number of intentions

Output: Generate utterances of $D$

1 foreach intention $z$ in the set of $N$ intentions do
   2 Draw $\beta_z \sim \text{Dirichlet}(\eta)$;
   3 end

4 foreach dialogue $d$ in $D$ do
   5 Draw $\theta \sim \text{Dirichlet}(\alpha)$;
   6 $\psi_1 \leftarrow 1$;
   7 foreach $i \leftarrow 2, \ldots, |d|$ do
      8 if beginning of a user utterance then
         9 Draw $\psi_i \sim \text{Bernoulli}(\epsilon)$;
      else
         10 $\psi_i \leftarrow 0$;
      end
   end
   13 foreach $i \leftarrow 1, \ldots, |d|$ do
      14 if $\psi_i = 0$ then
         15 $z_i \leftarrow z_{i-1}$;
      else
         16 Draw $z_i \sim \text{multinomial}(\theta)$;
      end
      18 Draw $w_i \sim \text{multinomial}(\beta_{z_i})$;
   end
   21 end

Moreover, we need to find the expected number of co-occurrence of a word $w$ with an intention $z$.

$$E(C_{z,w}) = |D| \sum_{i=1}^{N} \sum_{j=1}^{|d_i|} Pr(z_{i,j} = z, w_{i,j} = w|w_1, \ldots, w_{|d_i|})$$

where $d_i$ is the $i$th dialogue in the dialogue set $D$, and $w_{i,j}$ is the $j$th word of the $i$th dialogue.

2. In the maximization step, the maximum a posteriori (MAP) estimate for $\theta$ and $\beta$ is computed by the standard method of Lagrange multipliers [Bishop, 2006]:

$$\theta_{d,z} \propto E(C_{d,z}) + \alpha - 1$$
\[ \beta_{w,z} \propto E(C^{z,w}) + \eta - 1 \]

Note that, the vector \( \theta_z \) stores the probability of an intention \( z \):

\[ Pr(z|\theta) = \theta_z \quad (4.1) \]

And, the vector \( \beta_{w,z} \) stores the probability of an observation \( w \) given the intention \( z \):

\[ Pr(w|z,\beta) = \beta_{wz} \quad (4.2) \]

The parameter \( \epsilon \) denotes the dependency of the utterances on each other, i.e., how likely it is that two successive uttered utterances of the user have the same intention.

\[ \epsilon = \frac{\sum_{i=1}^{D} \sum_{j=1}^{d} Pr(\psi_{i,j} = 1|w_1, \ldots, w_{d_i})}{\sum_{i=1}^{D} N_{i,utt}} \]

where \( N_{i,utt} \) is the number of utterances in the dialogue \( i \).

Learning the parameters in HTMM can be done in a small computation time, using EM. This is a useful property, though EM suffers from local minima [Ortiz and Kaelbling, 1999] and the related work such as Griffiths and Steyvers [2004] proposed the Gibbs sampling method rather than EM. Ortiz and Kaelbling [1999], however, introduced methods for getting away from local minima, and also suggested that EM can be accelerated via some heuristics based on the type of the problem.

In HTMM, the special form of the transition matrix reduces the time complexity of the forward backward algorithm to \( O(TN) \), where \( T \) is the length of the chain, and \( N \) is the number of desired user intentions given to the algorithm [Gruber et al., 2007; Gruber and Popat, 2007]. The small computation time is particularly useful, as it allows the machine to update its model when it observes new data.

### 4.2.2 Learning intentions from SACTI-1 dialogues

In this section, we apply HTMM on SACTI-1 dialogues [Williams and Young, 2005], publicly available at: http://mi.eng.cam.ac.uk/projects/sacti/corpora/. SACTI stands for simulated ASR channel tourist information. It contains 144 dialogues between 36 users and 12 experts who play the role of the machine for 24 total tasks on this data set. The utterances are first recognized using a speech recognition error simulator, and then are sent to human experts for a response. There are four levels of ASR noise in SACTI-1 data: none, low, medium, and high noise. There is a total of 2048 utterances that we used for our experiments which have 817 distinct words.

Table 4.1 shows a dialogue sample from SACTI-1. The first line of the table shows the first user utterance, \( u_1 \). Because of ASR errors, this utterance is recognized as \( \tilde{u}_1 \). Then,
... 

$u_1$ yeah hello this is johan schmulka uh and i’m uh searching for a bar in this town can you may be tell me where the cafe blu is  

$\tilde{u}_1$ [hello this is now seven four bus and do you tell me where to cafe blu is]  

$m_1$ cafe blu is on alexander street  

$u_2$ oh um yeah how can i get to alexander street and where exactly is it i know there a shopping area on alexander street um  

$\tilde{u}_2$ [i am yeah i am at the alexander street and where is it was on a the center of alexander street]  

$m_2$ it is on the east side of alexander street so %um it’s %um just off middle road  

... 

Table 4.1: A sample from the SACTI-1 dialogues [Williams and Young, 2005].

$m_1$ is the actual machine utterance as a response to the user request recognized by the ASR in $\tilde{u}_1$. We applied HTMM as introduced in the previous section to learn possible user intentions in SACTI-1. In our experiments, we removed the machine responses from the dialogues in order to learn the user intentions based on the recognized user utterances. Nevertheless, since HTMM is an unsupervised learning method, we did not have to annotate the dialogues.

Table 4.2 shows the learned intentions from SACTI-1 data, using HTMM. The algorithm learns 3 user intentions which we named them respectively as:

1. visits,

2. transports,

3. foods.

Each intention is represented by its 20-top words with their probabilities. In Table 4.2, we have highlighted only the words which best represents each intention. These highlighted words are called keywords. To extract keywords, we avoided stop words such as the, a, an, to. For instance, the words hotel, tower, and castle are keywords which represent the user intentions for information necessary about visiting areas, i.e., visits.

Then, for each recognized user utterance $\tilde{u} = [w_1, \ldots, w_n]$, we define its subsequent
Table 4.2: The learned user intentions from the SACTI-1 dialogues.

\[ s = \arg \max_z Pr(w_1, \ldots, w_n | z) \]
\[ = \arg \max_z \prod_i Pr(w_i | z) \]
where $Pr(w_i|z)$ is already learned and stored in the parameter $\beta_{wz}$ according to Equation (4.2). The second equality in the equation, the product of probabilities, is due to the independency of words given a user intention.

User intentions have been previously suggested to be used as states of dialogue POMDPs [Roy et al., 2000; Zhang et al., 2001b; Matsubara et al., 2002; Doshi and Roy, 2007, 2008]. However, to the best of our knowledge, they have not been automatically extracted from real data. Here, we learn the user intentions based on unsupervised learning methods. This enables us to use raw data, with little annotation or preprocessing. In our previous work [Chinaei et al., 2009], we were able to learn 10 user intentions from SACTI-2 dialogues [Weilhammer et al., 2004], without annotating data or any preprocessing. In this paper, we showed cases where we can estimate the user intentions behind utterances when users did not use a keyword for an intention. In addition, we were able to learn the true intention behind recognized utterances that included wrong keywords or multiple keywords, possibly keywords of different learned intentions.

### 4.3 Learning the transition model

In the previous section, we learned states of the dialogue POMDP. In this section, we go through the second step of our descriptive Algorithm 1: extracting actions directly from dialogues and learning a maximum likelihood transition model.

In Section 3.1.1, we saw that a transition model is in the form of $T(s_1, a_1, s_2)$ where $T$ stores the probability of going to the state $s_2$ given performing the action $a_1$ in the state $s_1$. We learn a maximum likelihood transition model by performing the following counting:

$$T(s_1, a_1, s_2) = Pr(z'|z, a) = \frac{Count(z_1, a_1, z_2)}{Count(z_1, a_1)}$$ (4.4)

To do so, we extract the set of possible actions from the dialogue set. Then, the maximum probable intention (state) is assigned to each recognized utterance using Equation (4.3).

For instance, for the recognized utterances in the SACTI-1 example, we can learn the probability distribution of the intentions from Equation (4.2), denoted by $Pr$ in Table 4.3. Then, to calculate the state for each recognized utterance, we take the maximum probable state, using Equation (4.3). For instance, the user intention for $u_2$ is learned as $t$, i.e., *transports*.

Finally, the transition model can be learned using Equation (4.4). This is a maximum likelihood transition model. Figure 4.3 shows graphically that we use the maximum
Table 4.3: Learned probabilities of intentions for the recognized utterances in the SACTI-1 example.

<table>
<thead>
<tr>
<th>Utterance</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$ yeah hello this is johan schmulka uh and i’m uh searching for a bar in this town can you may be tell me where the cafe blu is</td>
<td>$Pr_1: 0.00$</td>
</tr>
<tr>
<td>$\tilde{u}_1$ [hello this is now seven four bus and do you tell me where to cafe blu is]</td>
<td></td>
</tr>
<tr>
<td>$u_2$ oh um yeah how can i get to alexander street and where exactly is it i know there a shopping area on alexander street um</td>
<td>$Pr_2: 0.99$</td>
</tr>
<tr>
<td>$\tilde{u}_2$ [i am yeah i am at the alexander street and where is it was on a the center of alexander street]</td>
<td></td>
</tr>
</tbody>
</table>

The transition model introduced in Equation (4.5) is similar to the user goal model for the factored transition model in Equation (3.7), proposed by Williams and Young [2007]; Williams [2006]. In contrast to the previous works, we learn such user model from dialogues, as described in Section 4.2.1, assign them to the recognized utterances by Equation (4.3), and then learn the smoothed maximum likelihood user model using Equation (4.5).
Figure 4.3: The maximum likelihood transition model is learned using the extracted actions, $a$, represented using the shaded square, and the learned states, $s$, represented in the light circles.

4.4 Learning observations and observation model

In this section, we go through the third step in the descriptive Algorithm 1. That is, reducing the observations significantly and learning the observation model. In this context, the definition of observations and observation model can be non-trivial. In particular, the time complexity for learning the optimal policy of a POMDP is double exponential to the number of observations [Cassandra et al., 1995]. In non-trivial domains such as ours, the number of observations is large. Depending on the domain, there can be hundreds or thousands of words which ideally should be used as observations. In this case, solving a POMDP with that many observations is intractable.

Therefore, in order to be able to apply POMDPs in such domains, we need to reduce the number of observations significantly. We learn an intention observation model based on HTMM. Figure 4.4 shows that the intention observations, denoted by $o$, are learned based on an unsupervised learning technique and added to the learned models. Before we propose the intention observation model, we introduce the keyword observation model.

4.4.1 Keyword observation model

For each state, this model uses the 1-top keyword which best represents the state. For instance, for SACTI-1 dialogues the 1-top keyword in Table 4.2 are the observations which include hotel, street, and restaurant. These observations can best represent the states: visits, transports, and foods, respectively. Moreover, an auxiliary observation,
Chapter 4. Dialogue POMDP model learning

The observations, $o$, are learned based on an unsupervised learning (UL) method, and are represented using the shaded circles.

which is called confusedObservation, is used, when none of the keyword observations occurs in a recognized user utterance. If an utterance includes more than one of the keyword observation, the confusedObservation is also used as the observation.

For the keyword observation model, we define a maximum likelihood observation model:

$$\Omega(o', a, s') = Pr(o'|a, s') = \frac{\text{Count}(a, s', o')}{\text{Count}(a, s')}$$

To make a more robust observation model, we apply smoothing to the maximum likelihood observation model for instance $\delta$ smoothing where $0 \leq \delta \leq 1$. We set $\delta$ to 1 to have add-1 smoothing:

$$\Omega(o', a, s') = Pr(o'|a, s') = \frac{\text{Count}(a, s', o') + 1}{\text{Count}(a, s') + \text{Count}(a, s', o')}$$

In the experiment of the observation models, in Section 6.2.2, the dialogue POMDP with the keyword observation model is called keyword POMDP.

### 4.4.2 Intention observation model

Given the recognized user utterance $\tilde{u} = [w_1, \ldots, w_n]$, the observation $o$ is defined in the same way as the state, i.e., the highest probable underlying intention in Equation (4.3). So the observation $o$ would be:

$$o = \arg\max_z \prod_{w_i} Pr(w_i|z)$$

(4.6)
Recall that $Pr(w_i|z)$ is learned and stored in the vector $\beta_{w_i}$ from Equation (4.2).

Notice that for the intention model, each state itself is the observation. As such, the set of observation is equivalent to the set of states. For instance, for SACTI-1 example the intention observations are $vo$, $to$, and $fo$ respectively for visits, transports, and foods states.

Similar to the keyword model, the intention observation model can be defined as:

$$\Omega(o',a,s') = Pr(o'|a,s') = \frac{\text{Count}(a,s',o')}{\text{Count}(a,s')}$$

Note that in the intention observation model, we essentially end up with a MDP model. This is because we use the highest probable intention as state and we use the highest probable intention as observation as well. So, we end up with a deterministic observation model, which is such as a MDP as discussed in Section 3.1.2. However, we can use a sort of smoothing to allow a small probability for other observations than the observation corresponding to the current state. In the experiment of the observation models, Section 6.2.2, we use the intention model without smoothing as the learned intention MDP model.

Additionally, we can estimate the intention observation model using the recognized utterances $\tilde{u}$ inside the training dialogue $d$, and using the vector $\beta_{wz}$ and $\theta_z$, reflected in Equation (4.2) and Equation (4.1), respectively. Assume that we want to estimate $Pr(o')$ in which $o'$ is drawn from Equation (4.6), then we have:

$$Pr(o') = \sum_w Pr(w,o')$$

$$= \sum_w Pr(w{o'}) Pr(o')$$

$$= \sum_w \beta_{wo'} \theta_{o'}$$

To estimate $Pr(o'|a,s')$, the multiplication in Equation (4.7) is performed only after visiting the action state pair $(a,s')$. Therefore, we use this calculation to learn the intention observation model. In the experiment of the observation models, Section 6.2.2, the dialogue POMDP with the intention observation model is called intention POMDP.

Atrash and Pineau [2010] proposed a Bayesian method of learning an observation model for POMDPs. Their observation model also draws from a Dirichlet distribution whose parameters are updated when the POMDP action matches with that of expert. More specifically, their proposed algorithm samples a few POMDPs of which only the observation models are different. Then, it learns the policy of each POMDP and go through a few runs by receiving an observation and performing the action of each POMDP. When the action of a POMDP matches with that of expert, observation model of that
POMDP is updated. The $n$ worst POMDP models are eliminated and then $n$ new POMDP models are sampled. This process continues until the algorithm is left with a few POMDPs in which the actions match highly with those of experts.

The work presented in Atrash and Pineau [2010] is different from ours as their work is a sample-based Bayesian method. That is, $n$ models are sampled and after updating each model, each POMDP model is solved, and the POMDP models are kept in which actions matched to the expert actions. The proposed observation models in this thesis, however, learns from expert/machine dialogues; it directly learns the observation model from dialogues and then learns the policy of the learned POMDP model.

As mentioned in Section 3.2.2, Png and Pineau [2011] proposed a Bayesian approach for updating the observation model of SmartWheeler dialogue POMDP. Similar to Ross et al. [2007, 2011], Png and Pineau [2011] used a Bayes-Adaptive POMDP for learning the observation model. More specifically, they considered a parameter for Dirichlet counts inside the POMDP state model. As such, when the POMDP updates its belief it also updates the Dirichlet counts which subsequently leads to the update of the observation model. As opposed to Png and Pineau [2011], we learned the model totally from SmartWheeler dialogues. Moreover, our idea of observations is based on intentions or keywords that is learned from dialogues, whereas observations in Png and Pineau [2011] is given/assumed.

In our previous work [Chinaei et al., 2012], we applied the two observation models on SACTI-1 and SmartWheeler dialogues. Our experimental results showed that the intention observation model outperforms the keyword observation model, significantly, based on accumulated mean rewards in simulation runs. In Chapter 6, we show the two learned models on SmartWheeler dialogues and present the results. In the following section, we go through the illustrative example on SACTI-1, and learn a dialogue POMDP by application of the proposed methods of this chapter on SACTI-1 dialogues.

4.5 Example on SACTI dialogues

We use the proposed methods in Section 4.2, Section 4.3, and Section 4.4 to learn a dialogue POMDP from SACTI-1 dialogues. First, we use the learned intentions in Table 4.2 as states of the domain. Based on the captured intentions, we defined 3 non-terminal states for the SACTI-1 machine as follows:

1. visits ($v$)
2. transports ($t$)
3. foods ($f$).
Moreover, we defined two terminal states:

4. success,

5. failure

The two terminal states are for dialogues which end successfully and unsuccessfully (respectively). The notion of successful or unsuccessful dialogue is defined by user. In SACTI-1, the user assigns the level of precision and recall of the received information, after finishing each dialogue. This is the only explicit feedback that we require to define the terminal states of the dialogue POMDP. A dialogue is successful if its precision and recall are above a predefined threshold.

The set of actions comes directly from the SACTI-1 dialogue set, and they include:

1. Inform,

2. Request,

3. GreetingFarewell,

4. ReqRepeat,

5. StateInterp,

6. IncompleteUnknown,

7. ReqAck,

8. ExplAck,

9. HoldFloor,

10. UnsolicitedAffirm,

11. RespondAffirm,

12. RespondNegate,

13. RejectOther,


For instance, GreetingFarewell is used for initiating or ending a dialogue, Inform is used for giving information for a user intention, ReqAck is used for the machine request for user acknowledgement; StateInterp is used for interpreting the intentions
...  
\begin{tabular}{|l|}
\hline
\textit{u}$_1$ & yeah hello this is johan schmulka uh and  
\quad i’m uh searching for a bar in this town  
\quad can you may be tell me where the cafe blu is  
\hline
\textit{\tilde{u}}$_1$ & [hello this is now seven four bus  
\quad and do you tell me where to cafe blu is]  
\hline
\textit{o}$_1$ & \textit{confusedObservation} (fo)  
\hline
\textit{a}$_1$: & \textit{Inform}(foods)  
\hline
\textit{m}$_1$ & cafe blu is on alexander street  
\hline
\textit{u}$_2$ & oh um yeah how can i get to alexander street and  
\quad where exactly is it i know there a shopping area  
\quad on alexander street um  
\hline
\textit{\tilde{u}}$_2$ & [i am yeah i am at the alexander street and  
\quad where is it was on a the center of alexander street]  
\hline
\textit{o}$_2$ & \textit{street} (to)  
\hline
\textit{a}$_2$: & \textit{Inform}(transports)  
\hline
\textit{m}$_2$ & it is on the east side of alexander street so  
\quad %um it’s %um just off middle road  
\hline
\end{tabular}

Table 4.4: Results of applying the two observation models on the SACTI-1 sample.

of user. Using such states and actions, the transition model of our dialogue POMDP was learned based on the method in Section 4.3.

The observations for SACTI-1 would be \textit{hotel, street, restaurant, confusedObservation, success, failure} in the case of keyword observation model, and the observations would be \textit{vo, to, fo, success, failure} in the case of intention observation model. Then, based on the proposed methods in Section 4.4, both keyword and intention observation models are learned. As mentioned in the previous section, the intention POMDP with the deterministic observation model is the intention MDP, which is used for the experiments of Chapter 5 and Chapter 6.

For our experiments, we used a typical reward model. Similar to previous work, we penalized each action in non-terminal states by -1, i.e., -1 reward for each dialogue turn [Williams and Young, 2007]. Moreover, actions in the success terminal state receive +50 as reward and actions in the failure terminal state receive -50 as reward.

Table 4.4 represents the sample from SACTI-1, introduced in Table 4.1, after applying the two observation models on the dialogues. The first user utterance is shown in \textit{u}$_1$. Note that \textit{u}$_1$ is hidden to the machine and is recognized as the line in \textit{\tilde{u}}$_1$. Then, \textit{\tilde{u}}$_1$
is reduced and received as the observation in $o_1$; if the keyword observation model is used the observation will be confusedObservation. This is because none of the keywords hotel, street, and restaurant occur in $\tilde{u}_1$. But, if the intention observation model is used then the observation inside parenthesis is used, i.e., fo which is an observation with high probability for foods state, and with small probability for visits and transports states.

The next line, $a_1$ shows the machine action in the form of dialogue acts. For instance, Inform(foods) is the machine dialogue act which is uttered by the machine as $m_1$, i.e., cafe blu is on alexander street. Next, the table shows $u_2$, $\tilde{u}_2$, $o_2$, and $a_2$. Note that in $o_2$, as opposed to $o_1$ in the case of keyword observation model, the keyword street occurs in the recognized utterance $\tilde{u}_2$.

### 4.5.1 HTMM evaluation

We evaluated HTMM for learning user intentions in dialogues. To achieve that, we measured the performance of the model on the SACTI data set based on the definition of perplexity similar to Blei et al. [2003]; Gruber et al. [2007]. For a learned topic model on a train data set, perplexity can be considered as a measure of on average how many different equally probable words can follow any given word. Therefore, it measures how difficult it is to estimate the words from the model. So, the lower the perplexity is, the better is the model.

Formally, the perplexity of a test dialogue $d$ after observing the first $k$ words can be drawn using the following equation:

$$
\text{perplexity} = \exp\left(-\frac{\log P_{\text{model}}(w_{k+1}, \ldots, w_{|d|}|w_1, \ldots, w_k)}{|d| - k}\right)
$$

We can manipulate the probability distribution in the equation above as:

$$
P_{\text{model}}(w_{k+1}, \ldots, w_{|d|}|w_1, \ldots, w_k) = \sum_{i=1}^{N} P_{\text{model}}(w_{k+1}, \ldots, w_{|d|}|z_i) P_{\text{model}}(z_i|w_1, \ldots, w_k)
$$

where $z_i$ is a user intention in the set of $N$ captured user intentions from the train set. Given a user intention $z_i$, the probability of observing $w_{k+1}, \ldots, w_{|d|}$ are independent of each other, so we have:

$$
P_{\text{model}}(w_{k+1}, \ldots, w_{|d|}|w_1, \ldots, w_k) = \sum_{i=1}^{N} \prod_{j=k+1}^{|d|} P_{\text{model}}(w_j|z_i) P_{\text{model}}(z_i|w_1, \ldots, w_k)
$$

To find out the perplexity, we learned the intentions for each test dialogue $d$ based on the first $k$ observed words in $d$, i.e., $\theta_{\text{new}} = P_{\text{model}}(z_i|w_1, \ldots, w_k)$ is calculated for each test
dialogue, whereas the vector $\beta$, which retains $Pr(w_j|z_i)$ (cf. Equation (4.2)), is learned from the training dialogues. We calculated the perplexity for 5% of the dialogues in data set and we used the 95% rest for training. Figure 4.5 shows the average perplexity after observing the first $k$ utterances of test dialogues. As the figure shows, the perplexity is reduced significantly when we observe new utterances.

At the end of Section 4.2.1 we mentioned that HTMM has a small computation time since it has a special form of the transition matrix [Gruber et al., 2007; Gruber and Popat, 2007]. Here we show the convergence rate of HTMM based on the convergence of log likelihood of data. Figure 4.6 shows the log likelihood of the observations for 30 iterations of the algorithm. We can see in the figure that the algorithm converges quite fast. For the given observations, the log likelihood is computed by averaging over possible intentions:

$$\hat{\theta}_{\text{MLE}} = \frac{\sum |D| |d_i|}{\sum j=1} \log \sum t=1 N Pr(w_i,j = w | z_{i,j} = z_t)$$

**Figure 4.5:** Perplexity trend with respect to increase of the number of observed user utterances.
Figure 4.6: Log likelihood of observations in HTMM as a function of the number of iterations.

4.5.2 Learned POMDP evaluation

We evaluated the learned intention POMDP from SACTI-1 dialogues, introduced in Section 4.2.2, using simulation runs. These results have been presented in our previous work [Chinaei and Chaib-draa, 2011]. The learned intention dialogue POMDP models from SACTI-1 consist of 3 non-terminal states and 2 terminal states, 14 actions, and 5 intention observations. We solved our POMDP models, using the ZMDP software available online at: http://www.cs.cmu.edu/~trey/zmdp/. We set a uniform distribution on the 3 non-terminal states, visits, transports, and foods, and set the discount factor to 0.90.

Based on simulation runs, we evaluated the robustness of the learned POMDP models to the ASR noise. There are four levels of ASR noise in SACTI data: none, low, medium, and high noise. For each noise level, we randomly took 24 available expert dialogues, calculated the average accumulated rewards for the experts from the 24 expert dialogues, and made a dialogue POMDP model from the 24 expert dialogues. Then, for each learned POMDP we performed 24 simulations and calculated their average accumulated rewards. In our experiments, we used the default simulation in the ZMDP software.

Figure 4.7 plots the average accumulated rewards as the noise level changes from 0 to 3 for none, low, medium, and high levels of noise (respectively). As the figure shows, the
dialogue POMDP models are robust to the ASR noise levels. That is, performance of the learned dialogue POMDPs decrease only slightly as the noise level increase. On the other hand, performance of experts decreases significantly, in particular at high level of noise. Note in Figure 4.7 that average accumulated mean reward for the experts is highest when there is no noise, and it is higher than the subsequent learned POMDPs. This is reasonable as the human expert can have best performance in the least uncertain conditions, i.e., when there is no noise.

Moreover, we evaluated the performance of the learned dialogue POMDPs as a function of expert dialogues (as training data), shown in Figure 4.8. Similar to the previous experiments, we calculated the average accumulated rewards for the learned POMDPs and for the experts from the subsequent expert dialogues. Overall, performance of the learned dialogue POMDPs is directly related to the number of expert dialogues and we find that more training data implies better performance.

Table 4.5 shows a sample from the learned dialogue POMDP simulation. The first action, $a_1$, is generated by dialogue POMDP, which is shown in the form of natural language in the following line, denoted by $m_1$. Then, the observation $o_2$ is generated by environment, $v_0$. For instance, the recognized user utterance could have been an utterance such as: $\tilde{u} : I \text{ would like a hour there museum first}$, and therefore its intention observation can be calculated using Equation (4.6). Notice that these results are only
Chapter 4. Dialogue POMDP model learning

Figure 4.8: Average rewards accumulated by the learned dialogue POMDPs with respect to the size of expert dialogues as training data.

Based on the dialogue POMDP simulation; where there exists neither user utterance nor machine’s utterance but only the simulated action and observations. Then, based on the received observation the POMDP belief, shown in $b_1$, is updated, using Equation (3.3). Based on belief $b_1$, the dialogue POMDP performs the next action, denoted by $a_2$.

In turns 3 to 5 shown in Table 4.5, we can see that the learned dialogue POMDP performs intuitively. In turn 3, the dialogue POMDP informs the user about transports, after receiving the observation to in turn 2 (the observation for transports). In $a_4$, the dialogue POMDP requests for acknowledgement that the user actually looks for transports, perhaps since it has already informed the user about transports in turn 3. After receiving the observation to in turn 4, and updating the belief, the dialogue POMDP informs the user again about transports in $a_5$.

4.6 Conclusions

In this chapter, we introduced methods for learning the dialogue POMDP states, transition model, observations and observation model, from recognized user utterances. In the intention-based dialogue domains in which the user intention is the dialogue state, an interesting problem is to learn the user intentions from unannotated user utterances.
To do so, first we studied HTMM, an unsupervised topic modeling approach that adds Markovian property to the LDA model. We then applied the HTMM method on dialogues to learn the set of user intentions and thus the probability distribution of user intentions for each recognized user utterance. We then made use of the learned user intentions as the dialogue POMDP states and learned a smoothed maximum likelihood transition model. Furthermore, we proposed two sets of observations: keyword and intention observations, automatically learned from dialogues, as well as their subsequent observation models.

Throughout this chapter, we applied the proposed methods on SACTI dialogues; we then evaluated the HTMM method for learning user intentions using SACTI dialogues, based on the definition of perplexity. Finally, we evaluated the learned intention dialogue POMDPs in simulation runs based on average accumulated rewards. The simulation results show that the learned intention dialogue POMDPs are robust to the ASR noise.

Building on the learned dialogue POMDP model components, in the next chapter, we propose two algorithms for learning the reward model based on IRL techniques.

### Table 4.5: A sample from SACTI-1 dialogue POMDP simulation.

<table>
<thead>
<tr>
<th>$a_1$:</th>
<th>GreetingFarewell</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1$:</td>
<td>How can I help you?</td>
</tr>
<tr>
<td>$o_2$:</td>
<td>vo</td>
</tr>
<tr>
<td>$b_1$:</td>
<td>$t:0.04814$ $v:0.91276$ $f:0.03909$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$a_2$:</th>
<th>INFORM(visits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_2$:</td>
<td>Here is information about visiting areas</td>
</tr>
<tr>
<td>$o_2$:</td>
<td>to</td>
</tr>
<tr>
<td>$b_2$:</td>
<td>$t:0.96732$ $v:0.00818$ $f:0.02449$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$a_3$:</th>
<th>INFORM(transports)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_3$:</td>
<td>Here is information about transportation</td>
</tr>
<tr>
<td>$o_3$:</td>
<td>to</td>
</tr>
<tr>
<td>$b_3$:</td>
<td>$t:0.99385$ $v:0.00031$ $f:0.00583$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$a_4$:</th>
<th>REQACK(transports)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_4$:</td>
<td>Are you looking for transportation</td>
</tr>
<tr>
<td>$o_4$:</td>
<td>to</td>
</tr>
<tr>
<td>$b_4$:</td>
<td>$t:0.94565$ $v:0.04833$ $f:0.00600$</td>
</tr>
</tbody>
</table>

| $a_5$: | INFORM(transports) |
Chapter 5

Reward model learning

5.1 Introduction

In Section 3.1, we introduced reinforcement learning (RL) as a technique for learning policy in stochastic/uncertain domains. In this context, RL works by optimizing a defined reward model in the (PO)MDP framework. In particular, choice of the reward model has been usually hand-crafted based on the domain expert intuition. However, it is evidently more convenient for the expert to demonstrate the policy. Thus, recently the inverse reinforcement learning (IRL) method is used to approximate the reward model that some expert agent appears to be optimizing.

Recall Figure 3.1 which showed the interaction between a machine and its environment. We present again the figure here, this time with more details in Figure 5.1. In this figure, circles represent learned models. The model denoted by POMDP includes the POMDP model components (without a reward model) which have been learned from introduced methods in Chapter 4. The learned POMDP together with action/observation trajectories are used in IRL to learn the reward model, denoted by R. Then, the learned POMDP and reward model are used in a POMDP solver to learn/update the optimal policy.

In this chapter, we introduce IRL and propose POMDP-IRL algorithms for the fourth step of the descriptive Algorithm 1: learning the reward model based on inverse reinforcement learning (IRL) techniques and using the learned POMDP model components. In this context, Ng and Russell [2000] proposed multiple IRL algorithms in the MDP framework that work by maximizing the sum of the margin between the policy of the expert (agent) and the intermediate candidate policies. These algorithms account for the case in which the expert policy is represented explicitly and the case where the expert policy is known only through observed expert trajectories.
Chapter 5. Reward model learning

IRL in POMDPs, in short POMDP-IRL, is particularly challenging due to the difficulty in solving POMDPs as discussed in Section 3.1.2. Recently, Choi and Kim [2011] proposed POMDP-IRL algorithms by extending MDP-IRL algorithms of Ng and Russell [2000] to POMDPs. In particular, Choi and Kim [2011] provided a general framework for POMDP-IRL by modeling the expert policy as a finite state controller (FSC) and thus using point-based policy iteration (PBPI) [Ji et al., 2007] as POMDP solver. The trajectory-based algorithms in Choi and Kim [2011] also required the FSC-based POMDP solvers (PBPI). In particular, they proposed a trajectory-based algorithm called max-margin between values (MMV) for the POMDP framework. Since such algorithms spent most of the time solving the intermediate policies, they suggested modifying the trajectory-based algorithms to be able to use other POMDP solvers such as Perseus [Spaan and Vlassis, 2005], etc.

In this chapter, we extend the trajectory-based MDP-IRL algorithm of Ng and Russell [2000] to POMDPs. We assume that the model components are known, similar to Ng and Russell [2000]; Choi and Kim [2011]. Fortunately, in dialogue management, the transition and observation models can be calculated from Wizard-of-Oz data [Choi and Kim, 2011] or a real system data, as mentioned in Section 1.1. In particular, in Chapter 4, we proposed methods for learning such components from data and showed the
illustrative example of learning the dialogue POMDP model components from SACTI-1 dialogues, collected in a Wizard-of-Oz setting [Williams and Young, 2005]. Then, the learned dialogue POMDP model together with expert dialogue trajectories can be used in IRL algorithms to learn a reward model for the expert policy.

In this context, IRL is an *ill-posed* problem. That is, there is not a single reward model that makes expert policy optimal, but infinitely many of them. We show this graphically in Figure A.1 in the appendix, through an experiment on a toy dialogue MDP. Since there are many reward models that makes the expert policy optimal, one approach is based on linear programming to find one of the possible solutions. The linear program constraints the set of possible reward models where the rewards are represented as a linear representation of dialogue features, and finds a solution among the limited set of solutions.

Note that in (PO)MDP-IRL the expert is assumed to be a (PO)MDP expert. That is, the expert policy is the policy that the underlying (PO)MDP framework optimizes. Similar to the previous work, we perform our IRL algorithm on (PO)MDP experts in this thesis.

In Section 5.2, we introduce the basic definitions of IRL. In this section, we also study in detail the main trajectory-based IRL algorithm for MDPs, introduced by Ng and Russell [2000]. We call this algorithm MDP-IRL. The material in Section 5.2 makes the foundation on which Section 5.3 is built. In particular, in Section 5.3.1 we propose a trajectory-based IRL algorithm for POMDPs, called POMDP-IRL-BT, which is an extension of the MDP-IRL algorithm of Ng and Russell [2000] for POMDPs. Then, in Section 5.3.2 we describe a point-based IRL algorithm for the POMDP framework, called PB-POMDP-IRL. In Section 5.4, we go through IRL related work, particularly for POMDPs. In Section 5.6, we revisit the SACTI-1 example; we apply the POMDP-IRL-BT and PB-POMDP-IRL algorithms on the learned dialogue POMDP from SACTI-1 (introduced in Section 4.5) and compare the results. Finally, we conclude this chapter in Section 5.7.

## 5.2 Inverse reinforcement learning in the MDP framework

In IRL, given an expert policy and an underlying MDP, the problem is to learn a reward model that makes the expert policy optimal. That is, given the expert policy, approximate a reward function for the MDP such that the optimal policy of the MDP includes the expert policy. In this section, we describe IRL for MDPs (MDP-IRL) using expert trajectories, represented as $(s_0, \pi_E(s_0), \ldots, s_{|S|-1}, \pi_E(s_{|S|-1}))$. To begin let
us introduce the following definitions:

- an expert reward model, denoted by $R^{\pi_E}$, is an unknown reward model for which the optimal policy is expert policy. We have the following definitions:
  - the expert policy, denoted by $\pi_E$, is a policy of the underlying MDP that optimizes the expert reward model $R^{\pi_E}$,
  - the value of the expert policy, denoted by $V^{\pi_E}$, is the value of the underlying MDP in which the reward model is the expert reward model $R^{\pi_E}$.

- a candidate reward model, denoted by $R$, is a reward model that could potentially be the expert reward model. We have the following definitions:
  - the candidate policy, denoted by $\pi$, is a policy of the underlying MDP that optimizes the candidate reward model $R$,
  - the value of the candidate policy, denoted by $V^\pi$, is the value of the candidate policy $\pi$ that optimizes the candidate reward $R$.

Then, IRL aims to find a reward model in which the expert’s policy is both optimal and maximally separated from other policies. To do this, some candidate reward models and their subsequent policies are generated from the expert’s behavior. The candidate reward model is approximated by maximizing the value of the expert policy with respect to all previous candidate policies. The new candidate reward model and policy are then used to approximate another new set of models. This process iterates until the difference in values of successive candidate policies is less than some threshold. The final candidate reward model is the solution to the IRL task.

Formally, we formulate the IRL problem as a MDP without a reward model, denoted by $\text{MDP}\setminus R = \{S,A,T_0,\gamma\}$, so that we can calculate the optimal policy of the MDP given any choice of candidate reward model. Having $t$ candidate policies $\pi_1, \ldots, \pi_t$, the next candidate reward is estimated by maximizing $d'$, the sum of the margins between value of expert policy and each learned candidate policy. Then, the objective function is as follows:

$$\text{maximize } d' = (v^{\pi_E} - v^{\pi_1}) + \ldots + (v^{\pi_E} - v^{\pi_t})$$ (5.1)

where $v^\pi$ is the vector representation for value function:

$$v^\pi = (v^\pi(s_0), \ldots, v^\pi(s_{|S|-1}))$$

and $v^\pi(s_i)$ is the value of state $s_i$ under policy $\pi$, which can be drawn from Equation (3.1). That is, we have:

$$v^\pi = r^\pi + \gamma T^\pi v^\pi$$ (5.2)
• $v^\pi$ is a vector of size $|S|$ in which $v^\pi(s) = V^\pi(s)$.

• $r^\pi$ is a vector of size $|S|$ in which $r^\pi(s) = R(s, \pi(s))$.

• $T^\pi$ is the transition matrix for policy $\pi$, that is a matrix of size $|S| \times |S|$ in which $T^\pi(s, s') = T(s, \pi(s), s')$.

Notice that in IRL it is assumed that the reward of any state $s$ can be represented as the linear combination of some features of state $s$, such as a feature vector defined as:

$$\phi = (\phi_1(s, a), \ldots, \phi_K(s, a))$$

where $K$ is the number of features and each feature $\phi_i(s, a)$ is a basis function for the reward model. The reward model can be shown as the multiplication of two vectors $\Phi^\pi$ and $\alpha$ as:

$$r^\pi = \Phi^\pi \alpha$$  \hspace{1cm} (5.3)

where $\alpha = (\alpha_1, \ldots, \alpha_K)$ are feature weights, and $\Phi^\pi$ is a matrix of size $|S| \times K$ consisting of state action features for policy $\pi$, defined as:

$$\Phi^\pi = \begin{pmatrix}
\phi(s_0, \pi(s_0))^T \\
\vdots \\
\phi(s_{|S|-1}, \pi(s_{|S|-1}))^T
\end{pmatrix}$$

For the expert policy $\pi_E$, the state action features become:

$$\Phi^{\pi_E} = \begin{pmatrix}
\phi(s_0, \pi_E(s_0))^T \\
\vdots \\
\phi(s_{|S|-1}, \pi_E(s_{|S|-1}))^T
\end{pmatrix}$$

We can manipulate Equation (5.2):

$$v^\pi = r^\pi + \gamma T^\pi v^\pi$$

$$v^\pi - \gamma T^\pi v^\pi = r^\pi$$

$$(I - \gamma T^\pi) v^\pi = r^\pi$$

$$v^\pi = (I - \gamma T^\pi)^{-1} r^\pi$$

Therefore, from the last equality we have:

$$v^\pi = (I - \gamma T^\pi)^{-1} r^\pi$$  \hspace{1cm} (5.4)

Using Equation (5.3) in Equation (5.4), we have:

$$\begin{align*}
  v^\pi &= (I - \gamma T^\pi)^{-1} \Phi^\pi \alpha \\
  v^\pi &= \mathbf{x}^\pi \alpha
\end{align*}$$  \hspace{1cm} (5.5)
where \( x^\pi \) is a matrix of size \(|S| \times K\) defined as:

\[
x^\pi = (I - \gamma T^\pi)^{-1} \Phi^\pi
\]  
(5.6)

Equation (5.5) shows that the vector of values \( v^\pi \) can be represented as multiplication of the feature weight vector \( \alpha \) and another vector \( x^\pi \).

Similar to Equation (5.5), for the expert policy \( \pi_E \), we have:

\[
v^{\pi_E} = x^{\pi_E} \alpha
\]  
(5.7)

where \( x^{\pi_E} \) is a matrix of size \(|S| \times K\) defined as:

\[
x^{\pi_E} = (I - \gamma T^{\pi_E})^{-1} \Phi^{\pi_E}
\]  
(5.8)

and \( T^{\pi_E} \) is a \(|S| \times |S|\) matrix where element \( T^{\pi_E}(s_i, s_j) \) is the probability of transiting from \( s_i \) to \( s_j \) with expert action \( \pi_E(s_i) \).

Therefore, both a candidate reward model and its subsequent candidate policy can be represented as multiplication of some feature function and the feature weights \( \alpha \) (see Equation (5.3) and Equation (5.5)). This enables us to solve Equation (5.1) as a linear program. Using Equation (5.5) and Equation (5.7) in Equation (5.1), we have:

\[
\text{maximize} \quad \alpha \left[ ((x^{\pi_E} - x^{\pi_1}) + \ldots + (x^{\pi_E} - x^{\pi_t})) \alpha \right] \\
\text{subject to} \quad -1 \leq \alpha_i \leq +1 \forall i, 1 \leq i \leq K
\]  
(5.9)

Having \( t \) candidate policies \( \pi_1, \ldots, \pi_t \), IRL estimates the next candidate reward by solving the above linear program. That is, IRL learns a new \( \alpha \) which represents a new candidate reward model, \( r = \Phi^{\pi_E} \alpha \). This new candidate reward has an “optimal policy” which is the new candidate policy \( \pi \).

Algorithm 7 shows the MDP-IRL algorithm introduced in [Ng and Russell, 2000]. This algorithm tries to find the expert reward model given an underlying MDP framework. The idea of this algorithm is that the value of expert policy is required to be higher than the value of any other policy under the same MDP framework. This is the maximization in Line 7 of the algorithm where \( v^{\pi_E} = x^{\pi_E} \alpha \) and \( v^{\pi_l} = x^{\pi_l} \alpha \) are the value of expert policy and the value of candidate policy \( \pi_l \), respectively. Notice that this algorithm maximizes the sum of the margins between the value of expert policy \( \pi_E \) and the value of other candidate policies \( \pi_l \).

Let’s go through Algorithm 7 in detail. The algorithm starts by randomly initiating values for \( \alpha \) to generate the initial candidate reward model \( R^1 \) in Line 1. Then, using dynamic programming for the MDP with the candidate reward model \( R^1 \), the algorithm finds policy of \( R^1 \), denoted by \( \pi_1 \). In Line 2, \( \pi_1 \) is used to construct \( T^{\pi_1} \) which is used
Algorithm 7: MDP-IRL: inverse reinforcement learning in the MDP framework, adapted from [Ng and Russell, 2000].

**Input:** $MDP\{R = \{S, A, T, \gamma\}$, expert trajectories in the form of $D = \{(s_n, \pi_E(s_n), s'_n)\}$, a vector of features $\phi = (\phi_1, \ldots, \phi_K)$, convergence rate $\epsilon$, and maximum iteration $maxT$

**Output:** Finds reward model $R$ where $R = \sum_i \alpha_i \phi_i(s, a)$ by approximating $\alpha = (\alpha_1, \ldots, \alpha_K)$

1. Choose the initial reward $R^1$ by randomly initializing feature weights $\alpha$;
2. Set $\Pi = \{\pi_1\}$ by finding $\pi_1$ using MDP with candidate reward model $R^1$ and value iteration;
3. Set $X = \{x^{\pi_1}\}$ by calculating $x^{\pi_1}$ using $T^{\pi_1}$ and Equation (5.6);
4. Calculate $x^{\pi_E}$ using $T^{\pi_E}$ and Equation (5.8);
5. for $t \leftarrow 1$ to $maxT$ do
   6. Find values for $\alpha$ by solving the linear program:
      maximize $d_t = \left[ (x^{\pi_E} - x^{\pi_1}) + \ldots + (x^{\pi_E} - x^{\pi_t}) \right] \alpha$
      subject to $|\alpha_i| \leq 1 \forall i 1 \leq i \leq K$;
   7. $R^{t+1} = \sum_i \alpha_i^t \phi_i(s, a)$;
   8. if $\max_i |\alpha_i^t - \alpha_i^{t-1}| \leq \epsilon$ then
      9. return $R^{t+1}$;
   10. else
      11. $\Pi = \Pi \cup \{\pi_{t+1}\}$ by finding $\pi_{t+1}$ using MDP with candidate reward model $R^{t+1}$ and value iteration;
      12. Set $X = X \cup \{x^{\pi_{t+1}}\}$ by calculating $x^{\pi_{t+1}}$ using $T^{\pi_{t+1}}$ and Equation (5.6);
   13. end
14. end
15. end

To calculate $x^{\pi_1}$ from Equation (5.6). Then, in Line 3, expert policy $\pi_E$ is used to construct $T^{\pi_E}$ which is used to calculate $x^{\pi_E}$ from Equation (5.8).

From Line 5 to Line 17, MDP-IRL goes through the iterations to learn expert reward model by solving the linear program in Line 7 with the constraints in Line 8. For instance, in the first iteration of MDP-IRL, using the linear programming above, the algorithm finds $\alpha$ which maximizes Equation (5.9). In Line 9, the learned vector values, $\alpha$, make a candidate reward model $R^2$ which introduces a candidate policy $\pi_2$ in Line 14. Then, in Line 15, $T^{\pi_2}$ is constructed for finding $x^{\pi_2}$ from Equation (5.6). The algorithm returns to Line 5 to repeat the process of learning a new candidate reward.
Chapter 5. Reward model learning

until convergence. In this optimization, we also constrain the value of the expert’s policy to be greater than that of other policies in order to ensure that the expert’s policy is optimal.

Note that in [Ng and Russell, 2000] there is a slight different algorithm for when expert policy is available in expert trajectories. The objective function for learning the reward model of expert maximizes sum of the margin between value of expert policy and that of other policies using a monotonic function \( f \). That is, the objective function in Ng and Russell [2000] is as follows:

\[
\text{maximize}_\alpha \quad d^t = \left[ f(v^{\pi_E} - v^{\pi_1}) + \ldots + f(v^{\pi_E} - v^{\pi_K}) \right] \tag{5.10}
\]

subject to \( |\alpha_i| \leq 1 \forall i \leq K \)

where Ng and Russell [2000] set \( f(x) = x \) if \( f(x) > 0 \), otherwise, \( f(x) = 2x \) to penalize the cases in which the value of expert policy is less than the candidate policy. The authors, selected 2 in \( f(x) = 2x \) since it had the least sensitivity in their experiments. The maximization in Equation (5.9) is similar to the one in Equation (5.10), particularly when \( f(x) = x \) for all \( x \).

Moreover, in Ng and Russell [2000] it is suggested to approximate the policy values using Monte Carlo estimator. Recall the definition of value function in MDPs, shown in Equation (3.1), defined as:

\[
V^\pi(s) = E_{s_t \sim T} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) | \pi, s_0 = s \right]
\]

Using \( M \) expert trajectory of size \( H \), the value function in MDPs can be approximated using Monte Carlo estimator:

\[
\hat{V}^\pi(s_0) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=0}^{H-1} \gamma^t R(s, a)
\]

The trajectory-based MDP-IRL algorithm in [Ng and Russell, 2000] has been extended to a model-free trajectory-based MDP-IRL algorithm, called LSPI-IRL, during the author’s internship at AT&T research labs in summer 2010 and during the author’s collaboration with AT&T research in 2011. In the LSPI-IRL algorithm, the candidate policies
are estimated using the LSPI (least square policy iteration) algorithm [Lagoudakis and Parr, 2003]. This algorithm is presented in the appendix, in Section A.2.

We then extended the trajectory-based MDP-IRL algorithm of [Ng and Russell, 2000] to a trajectory-based POMDP-IRL algorithm, called POMDP-IRL-BT, which is presented in Section 5.3.1.

5.3 Inverse reinforcement learning in the POMDP framework

In this section, we propose two IRL algorithms from expert trajectories in the POMDP framework. First in Section 5.3.1, we extend the MDP-IRL algorithm of Ng and Russell [2000] to POMDPs by approximating the value of expert policy and that of candidate policies (respectively Equation (5.7) and Equation (5.5)) for POMDPs. This is done by fixing the number of beliefs to the expert beliefs available in expert trajectories, and by approximating the expert belief transitions, i.e., the probability of transiting from one expert belief to another after performing an action. The algorithm is called POMDP-IRL-BT (BT for belief transitions). Then, in Section 5.3.2, we propose a point-based POMDP-IRL algorithm, called PB-POMDP-IRL.

5.3.1 POMDP-IRL-BT

We extend the trajectory-based MDP-IRL algorithm introduced in previous section to POMDPs. Our proposed algorithm, called POMDP-IRL-BT, considers the situation when expert trajectories are in form of \((a_1, o_1, \ldots, a_B, o_B)\), where \(B\) is the number of generated expert beliefs. Note that by application of the state estimator function in Equation (3.3), and an assumed belief \(b_0\), say the uniform belief, we can calculate expert beliefs \((b_0, \ldots, b_{B-1})\). Thus, expert trajectories can be represented as \((b_0, \pi_E(b_0), \ldots, b_{B-1}, \pi_E(b_{B-1}))\).

The POMDP-IRL-BT algorithm is similar to the MDP-IRL algorithm, described in Section 5.2, but instead of states we use the finite number of expert beliefs that occurred in expert trajectories. Moreover, we approximate a belief transition for expert beliefs in the place of the transition model in MDPs. More specifically, we approximate the value of the expert policy and the value of candidate policies by approximating Equation (5.7) and Equation (5.5), respectively, for POMDPs. Therefore, in IRL for POMDPs we maximize the margin:

\[
d' = (v_b^\pi - v_b^{\pi'}) + \ldots + (v_b^\pi - v_b^{\pi'})
\]
Chapter 5. Reward model learning

where \( v^{\pi} \) is an approximation of the value of the expert policy. This expert policy is based on the expert beliefs that occurred in expert trajectories. Moreover, each \( v^{\pi_t} \) is an approximation of value of the candidate policy \( \pi_t \) which is calculated by approximating expert belief transitions.

To illustrate these approximations, consider the value function for POMDPs shown in Equation (3.5). Using the vector representation, we can rewrite Equation (3.5) as:

\[
  v^\pi_b = r^\pi_b + \gamma P^\pi v^\pi_b
\]

where

- \( v^\pi_b \) is a vector of size \( B \): the number of expert beliefs in which \( v^\pi_b(b) = V^\pi(b) \) (from Equation (3.5)).
- \( r^\pi_b \) is a vector of size \( B \) in which \( r^\pi_b(b) = R(b, \pi(b)) \), where \( R(b, a) \) comes from Equation (3.4).
- \( P^\pi \) is a matrix of size \( B \times B \) that is the belief transition matrix for policy \( \pi \), in which:

\[
P^\pi(b, b') = \sum_{o' \in O} [Pr(o'|b, \pi(b)) \cdot ifClosest((SE(b, \pi(b), o'), b')]\)

where \( SE \) is the state estimator function in Equation (3.3) and \( ifClosest(b'', b') \) determines if \( b' \) is the closest expert belief to \( b'' \), the belief created as result of state estimator function. Formally, we define \( ifClosest(b'', b') \) as:

\[
  ifClosest(b'', b') = \begin{cases} 
    1, & \text{if } b' = \arg\min_{b_n} |b'' - b_n| \\
    0, & \text{otherwise}
  \end{cases}
\]

where \( b_n \) is one of the \( B \) expert beliefs that appeared within the expert trajectories.

\( P^\pi(b, b') \) is an approximate belief state transition model. It is approximated in three steps. First, the next belief \( b'' \) is estimated using the \( SE \) function. Second, the \( ifClosest \) function is used to find, \( b' \), the nearest belief that occurred within the expert trajectories. Finally, the transition probability between \( b \) and \( b' \) is updated using Equation (5.12). This avoids handling the excessive number of new beliefs created by the \( SE \) function. More importantly, this procedure supports the use of IRL on a fixed number of beliefs, such as expert beliefs from a fixed number of trajectories.

Figure 5.2 demonstrates how the the belief transition matrix is constructed for a candidate policy \( \pi \). Assume that the expert beliefs include only two belief points: \( b_0 \) and
Chapter 5. Reward model learning

$b_1$, as shown in Figure 5.2 top left. Then, the belief transition matrix is initialized to zero, as shown in Figure 5.2 top right. Starting from belief $b_1$, the action $\pi(b_1)$ is taken. If the observation $o_1$ is received then, using $SE$ function, the new belief $\tilde{b}_1$ is created, shown in Figure 5.2 middle left. The closest expert belief to $\tilde{b}_1$ is $b_0$, so the probability $Pr(o_1|b_1, \pi(b_1))$ is added to the transition from $b_1$ (the starting belief) to $b_0$ the landed belief, as shown in Figure 5.2 middle right. On the other hand, if the observation $o_2$ is received, then, using $SE$ function, the new belief $\tilde{b}_2$ is created, shown in Figure 5.2 bottom left. The closest expert belief to $\tilde{b}_2$ is $b_1$, so the probability $Pr(o_2|b_1, \pi(b_1))$ is added to the transition from $b_1$ (the starting belief) to $b_1$ the landed belief, as shown in Figure 5.2 bottom right.

We construct the rest of formulations similar to MDPs. The reward model, $R$, is represented using the vector of features $\phi$ so that each $\phi_i(s,a)$ is a basis function for the reward model. However, in POMDPs, we need to extend state features to beliefs. To do so, we define the vector $\phi(b,a)$ as: $\phi(b,a) = \sum_{s \in S} b(s) \phi(s,a)$. Then, matrix $\Phi_b^\pi$ is an $N \times K$ matrix of belief action features for policy $\pi$, defined as:

$$
\Phi_b^\pi = \begin{pmatrix}
\phi(b_0, \pi(b_0))^T \\
\vdots \\
\phi(b_{B-1}, \pi(b_{B-1}))^T
\end{pmatrix}
$$

For the expert policy $\pi_E$, we define $\Phi_b^\pi_E$ as:

$$
\Phi_b^\pi_E = \begin{pmatrix}
\phi(b_0, \pi_E(b_0))^T \\
\vdots \\
\phi(b_{B-1}, \pi_E(b_{B-1}))^T
\end{pmatrix}
$$

Formally, we define $r_b^\pi$ as:

$$
r_b^\pi = \Phi_b^\pi \alpha \quad (5.13)
$$

Similar to the MDP-IRL, we can manipulate Equation (5.11):

$$
\begin{align*}
\nu_b^\pi &= r_b^\pi + \gamma P^\pi \nu_b^\pi \\
\nu_b^\pi - \gamma P^\pi \nu_b^\pi &= r_b^\pi \\
(I - \gamma P^\pi)\nu_b^\pi &= r_b^\pi \\
\nu_b^\pi &= (I - \gamma P^\pi)^{-1} r_b^\pi
\end{align*}
$$

Therefore, from the last equality we have:

$$
\nu_b^\pi = (I - \gamma P^\pi)^{-1} r_b^\pi \quad (5.14)
$$

Using Equation (5.13) in Equation (5.14), we have:

$$
\begin{align*}
\nu_b^\pi &= (I - \gamma P^\pi)^{-1} \Phi_b^\pi \alpha \\
\nu_b^\pi &= \Phi_b^\pi \alpha
\end{align*}
$$

\text{(5.15)}
Figure 5.2: POMDP-IRL-BT illustration example.
where $x_b^\pi$ is a matrix of size $B \times K$ defined as:

$$x_b^\pi = (I - \gamma P^\pi)^{-1} \Phi_b^\pi$$  \hspace{1cm} (5.16)

Equation (5.15) shows that the vector of values $v_b^\pi$ can be represented as multiplication of the vector of feature weights $\alpha$ and the vector $x_b^\pi$.

We have a similar equation for the expert policy: $v_b^{\pi_E} = x_b^{\pi_E} \alpha$, where $x_b^{\pi_E}$ is a matrix of size $B \times K$ defined as:

$$x_b^{\pi_E} = (I - \gamma P^{\pi_E})^{-1} \Phi_b^{\pi_E}$$  \hspace{1cm} (5.17)

where $P^{\pi_E}$ is an $B \times B$ matrix where each element $P^{\pi_E}(b_i, b_j)$ is the probability of transiting from $b_i$ to $b_j$ with expert action $\pi_E(b_i)$.

Algorithm 8 shows POMDP-IRL-BT. Similar to MDP-IRL, this algorithm maximizes the sum of the margins between the expert policy $\pi_E$ and the candidate policies $\pi_t$ (Line 7 of Algorithm 8). The POMDP-IRL-BT algorithm is based on the belief transition model, as opposed to MDP-IRL which is based on transition of completely observable states.

Let’s go through Algorithm 8 in detail. The algorithm starts by randomly initiating values for $\alpha$ to generate the initial candidate reward model $R_1^t$ in Line 1. Then, the algorithm finds the policy of $R_1^t$, denoted by $\pi_1$, using a model-based POMDP algorithm such as point-based value iteration (PBVI) [Pineau et al., 2003]. In Line 3, $P^{\pi_1}$ is constructed, which is used to calculate $x_b^{\pi_1}$ from Equation (5.16). Then, in Line 4, the expert policy $\pi_E$ is used to construct $P^{\pi_E}$ which is used to calculate $x_b^{\pi_E}$ from Equation (5.17).

From Line 5 to Line 17, POMDP-IRL-BT iterates to learn the expert reward model by solving the linear program in Line 7 with the constraints shown in Line 8. The objective function of the linear program is:

$$\text{maximize}_\alpha \ d^t = \sum_{t=1}^t (x_b^{\pi_E} \alpha - x_b^{\pi_i} \alpha)$$

for all $t$ candidate policies learned so far up to iteration $t$, subject to the constraints $|\alpha_i| \leq 1 \ \forall i \ 1 \leq i \leq K$. So, it maximizes the sum of the margins between expert policy $\pi^*$ and other candidate policies $\pi_t$ (we have $t$ of them at iteration $t$). The rest is similar to the MDP-IRL. In this optimization, we also constrain the value of the expert’s policy to be greater than that of other policies in order to ensure that the expert’s policy is optimal.

As seen above, POMDP-IRL-BT approximates the expert policy value and the candidate policy values in POMDPs using the belief transition that is approximated in
Algorithm 8: POMDP-IRL-BT: inverse reinforcement learning in the POMDP framework using belief transition estimation.

**Input:** \( \text{POMDP} \setminus R = \{ S, A, T, \gamma, O, \Omega, b_0 \} \), expert trajectories in the form of
\( D = \{ (b_n, \pi_E(b_n), b'_n) \} \), a vector of features \( \phi = (\phi_1, \ldots, \phi_K) \), convergence rate \( \epsilon \), and maximum Iteration \( \text{maxT} \)

**Output:** Finds reward model \( R \) where \( R = \sum_i \alpha_i \phi_i(s, a) \) by approximating \( \alpha = (\alpha_1, \ldots, \alpha_K) \)

1. Choose the initial reward \( R^1 \) by randomly initializing feature weights \( \alpha \);
2. Set \( \Pi = \{ \pi_1 \} \) by finding \( \pi_1 \) using POMDP with candidate reward model \( R^1 \) and a PBVI variant POMDP solver;
3. Set \( X = \{ x_{b}^{\pi_1} \} \) by calculating \( x_{b}^{\pi_1} \) using \( P^{\pi_1} \) and Equation (5.16);
4. Calculate \( x_{b}^{*} \) from Equation (5.17);
5. for \( t \leftarrow 1 \) to \( \text{maxT} \) do
6.    Find values for \( \alpha \) by solving the linear program:
7.      maximize \( \alpha \left[ (x_{b}^{\pi_E} - x_{b}^{\pi_1}) + \ldots + (x_{b}^{\pi_E} - x_{b}^{\pi_t}) \right] \alpha \);  
8.      subject to \( -1 \leq \alpha_i \leq +1 \forall 1 \leq i \leq K \);  
9.      \( R^{t+1} = \sum_i \alpha_i \phi_i(s, a) \);  
10. if \( \max_i |\alpha_i^t - \alpha_i^{t-1}| \leq \epsilon \) then
11.    return \( R^{t+1} \);  
12.  else
13.    \( \Pi = \Pi \cup \{ \pi_{t+1} \} \) by finding \( \pi_{t+1} \) using POMDP with candidate reward model \( R^{t+1} \) and a PBVI variant POMDP solver;
14.    Set \( X = X \cup \{ x_{b}^{\pi_{t+1}} \} \) by calculating \( x_{b}^{\pi_{t+1}} \) using \( P^{\pi_{t+1}} \) and Equation (5.16);
15.  end
16. end

Equation (5.12). This approximation is done by first fixing the number of beliefs to expert beliefs. Moreover, after performing action \( a \) in a belief, we may end up to a new belief \( b'' \) (outside expert beliefs) which we map it to the closest expert belief.

In our previous work [Chinaei and Chaib-draa, 2012], we applied the POMDP-IRL-BT algorithm on POMDP benchmarks. Furthermore, we applied the algorithm on the dialogue POMDP learned from SmartWheeler (described in Chapter 6). The experimental results showed that the algorithm is able to learn a reward model that accounts for the expert policy. In Chapter 6, we apply the proposed methods in this thesis to learn a
dialogue POMDP from SmartWheeler dialogues; we also apply POMDP-IRL-BT on the learned dialogue POMDP and demonstrate the results.

5.3.2 PB-POMDP-IRL

In this section, we propose a point-based IRL algorithm for POMDPs, called PB-POMDP-IRL. The idea in this algorithm is that the value of new beliefs, i.e., the beliefs that are result of performing other policies than expert policy, are approximated using expert beliefs. Moreover, this algorithm constructs a linear program for learning a reward model for the expert policy by going through the expert trajectories and adding variables corresponding to the expert policy value and variables corresponding to the alternative policy values.

To understand the algorithm, we start by some definitions: we define each history $h$ as a sequence of observation action pairs of the expert trajectories denoted by $h = ((a_1, o_1), \ldots, (a_t, o_t))$. Moreover, we use $h_{ao}$ for the history of size $|h| + 1$ which includes the history $h$ followed by $(a, o)$. Then, we use $b_h$ to show the belief at the end of history $h$, which can be calculated using the State Estimator in Equation (3.3). We present State Estimator function again here:

$$b_{h_{ao}}(s') = SE(b_h, a, o) = Pr(s' | b_h, a, o) = \eta \Omega(a, s', o) \sum_{s \in S} b_h(s) T(s, a, s')$$

where $\eta$ is the normalization factor.

For instance, if $h = (a_1, o_1)$ then the belief at the end of history $h$, $b_h$ is calculated by the belief update function in Equation (3.3) and using $(a_1, o_1)$ and $b_0$ (usually a uniform belief) as the parameters. Similarly, if $h = ((a_1, o_1), \ldots, (a_t, o_t))$, the belief at the end of history is calculated by sequentially applying the belief update using $(a_i, o_i)$ and $b_{i-1}$ as the parameters.

The PB-POMDP-IRL algorithm is described in Algorithm 9. In our proposed algorithm, the value of new beliefs, i.e., the beliefs which are result of performing other policies (than expert policy), are approximated using expert beliefs. That is, given the belief $b_{h_{ao}}$ where $a \neq \pi_E(b_{h_{ao}})$, the value of $V^{\pi_E}(b_{h_{ao}})$ is approximated using expert histories $h'_{i}$ of the same size as $h_{ao}$, i.e., $|h'_{i}| = |h_{ao}|$. This approximation is demonstrated in Line 15 and Line 16 of the algorithm:

$$V^{\pi_E}(b_{h_{ao}}) = \sum_{i=0}^{n} w_i V(b_{h'_{i}})$$
such that $w_i$s follow:

$$b_{hao} = \sum_{i=0}^{n} w_i b_{h'_i}$$

Notice that due to the piecewise linearity of the optimal value function, this approximation corresponds to the true value if the expert policy in the belief state $b_{hao}$ is the same as the one in the belief states $b_{h'_i}$, which is used in the linear combination. This condition is more likely to be true when the beliefs $b_{h'_i}$ are closer to the approximated belief $b_{hao}$.

The algorithm also constructs a linear program for learning the reward model by going through expert trajectories and adding variables corresponding to the expert policy value and variables corresponding to alternative policy values. These variables are subject to the linear constraints that are subject to the Bellman equation (Line 20 and Line 23). In Line 20, the linear constraint for the expert policy value at end of history $h$ is added. This constraint is based on the Bellman Equation (3.5) which we present it again here:

$$V_\pi(b) = R(b, \pi(b)) + \gamma \sum_{o' \in O} Pr(o'|b, \pi(b)) V_\pi(b')$$

where here the rewards are presented as linear combination of state features:

$$R(s, a) = \sum_{i=1}^{k} \alpha_i \phi_i(s, a)$$

and $R(b, a)$ is defined as:

$$\sum_{s \in S} b(s) R(s, a)$$

So, the value of expert policy at end of history $h$ becomes:

$$V_{\pi_E}(b_h) = \left[ \sum_{s \in S} b_h(s) \sum_{i=1}^{k} \alpha_i \phi_i(s, \pi_E(b_h)) + \gamma \sum_{o \in O} Pr(o|b_h, \pi_E(b_h)) V_{\pi_E}(b_h \pi_E(b_h)) \right]$$

Similarly, in Line 23 the linear constraint for the alternative policy value at the end of history $h$ is added. Notice that an alternative policy is a policy that selects an action $a \neq \pi_E(b_h)$ and then follows the expert’s policy for the upcoming time-steps. This constraint is also based on the Bellman Equation (3.5). That is, the value of performing action $a$ at the belief $b_h$ where $a \neq \pi_E(b_h)$ and then following expert policy $\pi_E$ becomes:

$$V^a(b_h) = \sum_{s \in S} b_h(s) \sum_{i=1}^{k} \alpha_i \phi_i(s, a) + \gamma \sum_{o \in O} Pr(o|b_h, a) V_{\pi_E}(b_{hao})$$

Finally, in Line 25 we explicitly state that the expert policy value at any history $h$, $V_{\pi_E}(b_h)$ is higher than any alternative policy value, $V^a(b_h)$ where $a \neq \pi_E(b_h)$, by a margin $\epsilon_h^a$ that should be maximized in Line 29.

**Input:** A POMDP $\mathcal{M} = (S, A, O, T, \Omega, b_0, \gamma)$, expert trajectories $D$ in the form of $a_1^m o_1^m \ldots a_{t-1}^m o_{t-1}^m a_t^m, t \leq H$

**Output:** Reward weights $\alpha_i \in \mathbb{R}$;

1. Extract the human’s policy $\pi_E$ from the trajectories;
2. Initialize the set of variables $V$ with the weights $\alpha_i$;
3. Initialize the set of linear constraints $C$ with
   $$\{\forall (s, a) \in S \times A : R_{\min} \leq \sum_{i=1}^{k} \alpha_i \phi_i(s, a) \leq R_{\max}\}$$
4. for $t \leftarrow H$ to 1 do
   5. foreach $h \in D$, such that $h$ is a trajectory of length $t$, do
     6. Calculate $b_h$, the belief state at the end of trajectory $h$;
     7. foreach $(a, o) \in A \times O$ do
        8. Add the variable $V_{\pi_E}(b_{hao})$ to $V$;
        9. if $hao \notin D$ and $t = H$ then
           10. Add the constraint $V_{\pi_E}(b_{hao}) = 0$ to the set $C$
           end
        11. if $hao \notin D$ and $t < H$ then
           12. Let $b_{hao}$ be the belief corresponding to the trajectory $hao$;
           13. Calculate the belief states $b_{h_t}$ corresponding to the trajectories in $D$ of length $t+1$;
           14. Find a list of weights $w_i$ such that $b_{hao} = \sum_{i=0}^{n} w_i b_{h_t}$;
           15. Add to $C$ the constraint $V_{\pi_E}(b_{hao}) = \sum_{i=0}^{n} w_i V(b_{h_t})$;
           16. /* $V_{\pi_E}(b_{hao})$ is approximation of $\pi_E$ value at the belief corresponding to the trajectory $hao$ */
           end
        17. Add the variable $V_{\pi_E}(b_h)$ to $V$;
        18. /* $V_{\pi_E}(b_h)$ is $\pi_E$ value at $b_h$ */
        19. Add to $C$ the constraint $V_{\pi_E}(b_h) =$
          $$\left[\sum_{s \in S} b_h(s) \sum_{i=1}^{k} \alpha_i \phi_i(s, \pi_E(b_h)) + \gamma \sum_{o \in O} Pr(o|b_h, \pi_E(b_h)) V_{\pi_E}(b_{h\pi_E(b_h)o})\right]$$
        20. foreach $a \in A$ do
           21. Add the variable $V^a(b_h)$ to $V$;
           22. /* $V^a(b_h)$ is the value of the alternative policy that chooses $a$ after the trajectory $h$ */
           23. Add to $C$ the constraint $V^a(b_h) =$
             $$\sum_{s \in S} b_h(s) \sum_{i=1}^{k} \alpha_i \phi_i(s, a) + \gamma \sum_{o \in O} Pr(o|b_h, a) V_{\pi_E}(b_{hao})$$
           24. Add the variable $c_h^a$ to the set $V$;
           25. Add to $C$ the constraint $V_{\pi_E}(b_h) - V^a(b_h) \geq c_h^a$;
           end
     end
   end
   26. maximize $\sum_{h \in H} \sum_{a \in A} c_h^a$ subject to the constraints of set $C$;
5.3.3 PB-POMDP-IRL evaluation

In our previous work [Boularias et al., 2010], we evaluated the PB-POMDP-IRL performance as the ASR noise level increases. The results are shown in Table 5.1. We applied the algorithm on four dialogue POMDPs learned from SACTI-1 dialogues with four levels of noise none, low, medium, and high, respectively, as described in Section 4.5.2. Our experimental results showed that the PB-POMDP-IRL algorithm is able to learn a reward model for human expert policy. Note that SACTI dialogues have been collected in a Wizard-of-Oz setting. The results also show that the algorithm performs better in the lower noise levels (none and low) than in higher noise levels (medium and high). In Section 5.6, we compare the PB-POMDP-IRL algorithm to the POMDP-IRL-BT algorithm on SmartWheeler learned POMDP actions.

<table>
<thead>
<tr>
<th>Noise level</th>
<th>non</th>
<th>low</th>
<th>med</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>HC reward matches</td>
<td>339-24%</td>
<td>327-23%</td>
<td>375-26%</td>
<td>669-47%</td>
</tr>
<tr>
<td>Learned reward matches</td>
<td>869-61%</td>
<td>869-61%</td>
<td>408-28%</td>
<td>387-27%</td>
</tr>
</tbody>
</table>

Table 5.1: Number of matches for hand-crafted reward POMDPs, and learned reward POMDPs, w.r.t. 1415 human expert actions.

5.4 Related work

Inverse reinforcement learning has been mostly developed in the MDP framework. In particular, in Section 5.2, we studied the basic trajectory-based MDP-IRL algorithm, proposed by Ng and Russell [2000]. Later on, Abbeel and Ng [2004] introduced an apprenticeship learning algorithm via IRL, which aims to find a policy which is close to the expert policy. That is, a policy whose feature expectations is close to that of expert policy. The feature expectations are derived from the MDP value function in Equation (3.1), which we present it again here:

\[ V^\pi(s) = E_{s \sim T} \left[ \sum_{t=0}^{\infty} \gamma^t R(s, \pi(s)|s_0) \right] \tag{5.18} \]

\[ = E_{s \sim T} \left[ \sum_{t=0}^{\infty} \gamma^t \alpha^T \phi(s, \pi(s)|s_0) \right] \]

\[ = \alpha^T E_{s \sim T} \left[ \sum_{t=0}^{\infty} \gamma^t \phi(s, \pi(s)|s_0) \right] \]

\[ = \alpha^T \mu(\pi) \]

where the second equality is because the reward model is represented as the linear combination of features, similar to MDP-IRL, we have \( R(s, a) = \alpha \phi(s, a) \).
From Equation (5.18), we can see:

$$\mu(\pi) = E_{s\sim T} \left[ \sum_{t=0}^{\infty} \gamma^t \phi(s, \pi(s)) | s_0 \right]$$

in which $\mu(\pi)$ is the vector of expected discounted feature values $\mu(\pi)$, i.e., feature expectations. By comparing the definition of feature expectation $\mu(\pi)$ to the vector $x^\pi$ appearing in Equation (5.5), we learn that the vector $x^\pi$ is an approximation for feature expectation.

Then, the apprenticeship learning problem is reduced to the problem of finding a policy whose feature expectation is close to the expert policy feature expectation. This is done by learning a reward model as an intermediate step. Notice that in apprenticeship learning the learned reward model is not necessarily the correct underlying reward model [Abbeel and Ng, 2004]; as the objective in the algorithm is finding the reward model for the policy that has an approximate feature expectation close to the expert policy feature expectation.

In the POMDP framework, as mentioned in Section 5.1, Choi and Kim [2011] provided a general framework for IRL in POMDPs by assuming that expert policy is represented in the form of a FSC (finite state controller), and thus using a FSC-based POMDP solver called PBPI (point-based policy iteration) [Ji et al., 2007]. Similar to the trajectory-based algorithms introduced in this chapter, Choi and Kim [2011] proposed trajectory-based algorithms for learning the POMDP reward models (besides their proposed analytical-based algorithms). In particular, they proposed a trajectory-based algorithm called MMV (max-margin between values) described as follows.

The MMV algorithm is similar to the MDP-IRL algorithm, introduced in Section 5.2, which works given the MDP model and expert trajectories. In particular, Choi and Kim [2011] used an objective function for maximizing the sum of the margin between expert policy and other candidate policies using a monotonic function $f$, similar to Ng and Russell [2000] (cf. end of Section 5.2). Moreover, the policy values are estimated using the Monte Carlo estimator using expert trajectories. Recall the definition of value function in POMDPs, shown in Equation (3.5), defined as:

$$V^\pi(b) = E_{b_t \sim SE} \left[ \sum_{t=0}^{\infty} \gamma^t R(b_t, \pi(b_t)) | \pi, b_0 = b \right]$$

Using an expert trajectory of size $B$, the value of expert policy can be estimated using
the Monte Carlo estimator as:

\[ \hat{V}^{\pi_E}(b_0) = R(b_0, \pi_E(b_0)) + \ldots + R(b_{B-1}, \pi(b_{B-1})) \]

\[ = \sum_{t=0}^{B-1} \gamma^t R(b_t, a_t) \]

\[ = \alpha^T \sum_{t=0}^{B-1} \gamma^t \phi(b_t, a_t) \]

where the last equality comes from the reward model representation using features, shown in Equation (5.13).

Similar to the trajectory-based MMV algorithm of Choi and Kim [2011], we used the POMDP beliefs that appeared in the expert trajectories. In contrast to the FSC-based representation used in Choi and Kim [2011], we used the belief point representation. Furthermore, instead of approximating the policy values using the Monte Carlo estimator, we approximated the policy values by approximating the belief transition matrix in Equation (5.12).

In order to compare the belief transition estimation to the Monte Carlo estimation, we implemented the Monte Carlo estimator in Equation (5.19) for estimation of policy values in Line 7 of Algorithm 8, and used the Perseus software [Spaan and Vlassis, 2005] as the POMDP solver. This new algorithm is called POMDP-IRL-MC (MC for the Monte Carlo estimator) and described as follows.

### 5.5 POMDP-IRL-MC

Estimating policy values can be inaccurate, in both the introduced methods: the Monte Carlo estimator as well as the belief transition approximation, proposed in Equation (5.12) (in the POMDP-IRL-BT algorithm). This is because the number of expert trajectories is small compared to the infinite number of possible belief points. In order to compare the Monte Carlo estimation to the belief transition estimation, we implemented the Monte Carlo estimator in Equation (5.19) for estimation of policy values in Line 7 of Algorithm 8, and used the Perseus software [Spaan and Vlassis, 2005] as the POMDP solver. This new algorithm is called POMDP-IRL-MC which is similar to the MMV algorithm of Choi and Kim [2011], described in the previous section.

The deference between the MMV algorithm of Choi and Kim [2011] and POMDP-IRL-MC is the policy representation and consequently the POMDP solver. As mentioned above, Choi and Kim [2011] used FSC representation in their MMV algorithm and thus using PBPI, an FSC-based POMDP solver [Ji et al., 2007]. In POMDP-IRL-MC, however, we used belief point representation and thus used, Perseus, a point-based POMDP solver [Spaan and Vlassis, 2005] (similar to our POMDP-IRL-BT algorithm,
In Section 5.3.1, we proposed the POMDP-IRL-BT algorithm to the POMDP-IRL-MC in terms of solution quality and scalability.

### 5.6 POMDP-IRL-BT and PB-POMDP-IRL performance

In this section, we show the example of IRL on the learned dialogue POMDP from SACTI-1, introduced in Section 4.5. In particular, we apply POMDP-IRL-BT (introduced in Section 5.3.1), and PB-POMDP-IRL (introduced in Section 5.3.2) for learning the reward model of our example dialogue POMDP learned from SACTI-1. Recall the learned intention dialogue POMDP from SACTI-1. The POMDP model consists of 5 states, 3 non-terminal states for visits, transports, and foods intentions, as well as two terminal states success and failure. The POMDP model also includes 14 actions, 5 intention observations, and the learned transition and observation models. The learned SACTI-1 specification for IRL experiments, of this section, are described in Table 5.2.

As mentioned in Section 5.1, for the purpose of POMDP-IRL experiments, we consider expert policy as a POMDP policy similar to the previous works [Ng and Russell, 2000; Choi and Kim, 2011]. For the expert reward model, we assumed the reward model introduced in Section 4.5. That is, the reward model which penalizes each action in non-terminal states by -1. Moreover, any action in the success terminal state receives +50 as reward, and any action in the failure terminal state receives -50 as reward. Then, we solved the POMDP model to find the optimal policy and assumed it as the expert policy to generate 10 trajectories. Each trajectory is generated from the initial belief and by performing the expert action. After receiving an observation the expert belief is updated and the next action is performed. The trajectory ends when reaching one of the two terminal states. The 10 generated trajectories were then used in our two fold cross validation experiments.

We applied the POMDP-IRL-BT and PB-POMDP-IRL algorithms on the SACTI-1 dialogue POMDP using state-action-wise features in which there is an indicator function for each state-action pair. Since there are 5 states and 14 actions in the example dialogue POMDP, the size of features equals 70 = 5 × 14. To solve each POMDP model, we used the Perseus solver which is a PBVI (point-based value iteration) solver [Spaan and Vlassis, 2005]. As stated in Section 3.1.4.4, PBVI solvers are approximate solvers that use a finite number of beliefs for solving a POMDP model. We set the solver to use 10,000 random samples for solving the optimal policy of each candidate reward. The other parameter is max-time for execution of the algorithm, which is set to 1000.

The two fold cross validation experiments are done as follows. We randomly selected 5
Table 5.2: The learned SACTI-1 specification for IRL experiments.

| Problem   | $|S|$ | $|A|$ | $|O|$ | $\gamma$ | $|\phi|$ | trajectories |
|-----------|-----|-----|-----|--------|-------|------------|
| SACTI-1   | 5   | 14  | 5   | 0.90   | 70    | 50         |

Table 5.3: POMDP-IRL-BT and PB-POMDP-IRL results on the learned POMDP from SACTI-1: Number of matched actions to the expert actions.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># of matched actions</th>
<th>matched-percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>POMDP-IRL-BT</td>
<td>42</td>
<td>84%</td>
</tr>
<tr>
<td>PB-POMDP-IRL</td>
<td>29</td>
<td>58%</td>
</tr>
</tbody>
</table>

trajectories from the 10 expert trajectories, introduced above, for training and the rest of 5 trajectories for testing. Then we tested POMDP-IRL-BT and PB-POMDP-IRL. For each algorithm experiment, the algorithm was used to learn a reward model for the expert trajectories using the training trajectories. Then the learned policy, i.e., the policy of the learned reward model, was applied on the testing trajectories. Finally, we calculated the number of learned actions that matched to the expert actions on the testing trajectories, and they were added up for the two folds to make the cross validation experiments complete.

The experimental results are shown in Table 5.3. The results show that POMDP-IRL-BT significantly outperforms PB-POMDP-IRL. More specifically, the POMDP-IRL-BT algorithm was able to learn a reward model that matched with 42 actions out of 50 actions in the data set. That is, the policy of the learned reward model was equal to the expert policy for 84% of the beliefs. On the other hand, the learned policy using PB-POMDP-IRL matched to 29 actions out of the 50 actions in the data set, i.e., 58% match. Thus, in the next chapter, for learning the reward model, we apply POMDP-IRL-BT on the learned dialogue POMDP from SmartWheeler.

5.7 Conclusions

In this chapter, we first introduced IRL for learning the reward model of expert policy in the MDP framework. In particular, we studied MDP-IRL algorithm of [Ng and Russell, 2000], the basic IRL algorithm in the MDP framework. Then, we proposed two IRL algorithms in the POMDP framework: POMDP-IRL-BT and PB-POMDP-IRL.

The proposed POMDP-IRL-BT algorithm is similar to the MDP-IRL algorithm. That is, it maximizes sum of the margin between the expert policy and other intermediate candidate policies. Moreover, instead of states we used belief states and the optimization is performed only on the expert beliefs, rather than all possible beliefs, using an
approximated belief transition model. On the other hand, the idea in the proposed PB-POMDP-IRL algorithm is that the value of new beliefs, i.e., the beliefs that are result of performing other policies than expert policy, are linearly approximated using expert belief values. We then revisited the learned intention POMDP from SACTI-1 and applied the two proposed POMDP-IRL algorithms on it. The result of the experiments showed that POMDP-IRL-BT significantly outperforms PB-POMDP-IRL.

Learning the reward model from expert dialogues makes our descriptive Algorithm 1 complete. In the following chapter, we show the application of our proposed methods on healthcare dialogue management.
Chapter 6

Application on healthcare dialogue management

6.1 Introduction

In this chapter, we show the application of our proposed methods on healthcare dialogue management. That is, we use the methods in this thesis to learn a dialogue POMDP from real dialogues of an intention-based dialogue domain (cf. Chapter 1), known as SmartWheeler [Pineau et al., 2011]. The SmartWheeler project aims to build an intelligent wheelchair for persons with disabilities. In particular, SmartWheeler aims to minimize the physical and cognitive load required in steering it. This project has been initiated in 2006, and a first prototype, shown in Figure 6.1, was built in-house at McGill’s Center for Intelligent Machines.

We used the dialogues collected by SmartWheeler to develop a dialogue POMDP learned primarily from data. The data includes eight dialogues with healthy users and nine dialogues with target users of SmartWheeler [Pineau et al., 2011]. The dialogues with target users, who are the elderly, are somehow more noisy than the ones with healthy users. More specifically, the average word error rate (WER) equals 13.9% for the healthy user dialogues and 18.5% for the target user dialogues. In order to perform our experiments on a larger amount of data, we used all the healthy and target user dialogues. In total, there are 2853 user utterances and 422 distinct words in the SmartWheeler dialogues.

Table 6.1 shows a sample of SmartWheeler dialogues captured for training the dialogue POMDP model components. The first line denoted by $u_1$ shows the true user utterance, that is the one which has been extracted manually from user audio recordings. The following line denoted by $\tilde{u}_1$ is the recognized user utterances by ASR. Finally, the line denoted by $a_1$ shows the performed action in response to the ASR output at the time.
Notice that the true user utterance is not observable to SmartWheeler, and thus it requires to perform the action based on the recognized utterance by ASR. That is, for each dialogue utterance recognized by ASR, the machine aims to estimate the user intention and then to perform the best action that satisfies the user intention. The recognized utterance by ASR is not however reliable for decision making. For instance, the first utterance, $u_1 : \text{[turn right a little]}$, shows the true user utterance. The ASR output for this utterance is, 

\[ \tilde{u}_1 : \text{[10 writer little]}. \]

As such, the action performed by SmartWheeler at this dialogue turn is, the *general query action* $u_1 : \text{PLEASE REPEAT YOUR COMMAND}$. The query action, is the SmartWheeler action for getting more information. For instance, in the example in Table 6.1, when SmartWheeler receives the second ASR output [10 writer little], it performs a general query action to get more information before it performs the right action for the user intention, i.e., TURN RIGHT A LITTLE.
Table 6.1: A sample from the SmartWheeler dialogues [Pineau et al., 2011].

The list of all SmartWheeler actions are shown in Table 6.2. Each action is the right action of one state (the user intention for a specific command). So, ideally, there should be 24 states for SmartWheeler dialogues (There are 24 actions other than the general query action). However, in the next section we see that we only learned 11 of the states, mainly because of number of dialogues. That is, not all of the states appeared in the data frequently enough. There are also states that do not appear in dialogues at all.

In this chapter, in Section 6.2, we learn a dialogue POMDP from SmartWheeler. First in Section 6.2.1, we learn a keyword POMDP and an intention POMDP (without the reward model) from SmartWheeler noisy dialogues based on the introduced methods in Chapter 4. Then in Section 6.2.2, we compare the intention POMDP performance to the keyword POMDP performance.

In Section 6.3, we go through set of experiments for IRL in SmartWheeler. First in Section 6.3.1, we learn a set of features for SmartWheeler, called keyword features. Then in Section 6.3.2, we use them for MDP-IRL application on the learned dialogue MDP from SmartWheeler. Then, in Section 6.3.3 we experiment POMDP-IRL-BT on the SmartWheeler learned intention POMDP using the keyword features. In Section 6.3.4, we compare POMDP-IRL-BT and POMDP-IRL-MC, introduced in Section 5.5, using the learned intention POMDP from SmartWheeler. Finally, we conclude this chapter in Section 6.4.
6.2 Dialogue POMDP model learning for SmartWheeler

We learned the possible user intentions in SmartWheeler dialogue based on the HTMM method as explained in Section 4.2.1. To do so, we preprocessed the dialogues to remove stop words such as determiners and auxiliary verbs. Then, we learned the user intentions for the SmartWheeler dialogues. Table 6.3 shows the learned user intentions with their four top words. Most of the learned intentions show a specific user command:

\[ i_1 : \text{move forward little}, \ i_2 : \text{move backward little}, \ i_3 : \text{turn right little}, \]
\[ i_4 : \text{turn left little}, \ i_5 : \text{follow left wall}, \ i_6 : \text{follow right wall}, \]
\[ i_8 : \text{go door}, \ \text{and} \ i_{11} : \text{stop}. \]

Table 6.2: The list of the possible actions, performed by SmartWheeler.
<table>
<thead>
<tr>
<th>intention 1</th>
<th>intention 2</th>
<th>intention 3</th>
<th>intention 4</th>
<th>intention 5</th>
<th>intention 6</th>
<th>intention 7</th>
<th>intention 8</th>
<th>intention 9</th>
<th>intention 10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>forward</strong></td>
<td><strong>backward</strong></td>
<td><strong>right</strong></td>
<td><strong>left</strong></td>
<td><strong>right</strong></td>
<td><strong>top</strong></td>
<td><strong>stop</strong></td>
<td><strong>bottom</strong></td>
<td><strong>stop</strong></td>
<td><strong>top</strong></td>
</tr>
<tr>
<td>0.180</td>
<td>0.380</td>
<td>0.209</td>
<td>0.189</td>
<td>0.279</td>
<td>0.143</td>
<td>0.942</td>
<td>0.022</td>
<td>0.088</td>
<td>0.131</td>
</tr>
<tr>
<td>move</td>
<td>drive</td>
<td>turn</td>
<td>go</td>
<td>wall</td>
<td>stop</td>
<td>word</td>
<td>door</td>
<td>for</td>
<td>speed</td>
</tr>
<tr>
<td>0.161</td>
<td>0.333</td>
<td>0.171</td>
<td>0.171</td>
<td>0.212</td>
<td>0.131</td>
<td>0.080</td>
<td>0.289</td>
<td>0.088</td>
<td>0.058</td>
</tr>
<tr>
<td>little</td>
<td>little</td>
<td>little</td>
<td>go</td>
<td>follow</td>
<td>follow</td>
<td>speed</td>
<td>forward</td>
<td>move</td>
<td>set</td>
</tr>
<tr>
<td>0.114</td>
<td>0.109</td>
<td>0.131</td>
<td></td>
<td>0.197</td>
<td>0.098</td>
<td></td>
<td>0.071</td>
<td>0.161</td>
<td>0.054</td>
</tr>
<tr>
<td>drive</td>
<td>top</td>
<td>bit</td>
<td></td>
<td>left</td>
<td>person</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.081</td>
<td>0.017</td>
<td>0.074</td>
<td></td>
<td>0.064</td>
<td>0.096</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3: The learned user intentions from the SmartWheeler dialogues.

There are two learned intentions that loosely represent a command:

\[ i_9 : \text{set speed} \quad \text{and} \quad i_{10} : \text{follow person}. \]

And, there is a learned intention that represent two commands:

\[ i_7 : \text{turn degree right/left}. \]

Table 6.4 shows results of HTMM application on SmartWheeler for the example shown in Table 6.1. The line denoted by \( u \) is the true user utterance, manually extracted by listening to the dialogue recordings. Then, \( \hat{u} \) is the recognized user utterance by ASR. For each recognized utterance, the following three lines show the probability of each user intention, denoted by \( Pr \). Finally, the last line, denoted by \( a \), shows the performed action by SmartWheeler.

For instance, the second utterance shows that the user actually uttered \textit{turn right a little}, but it is recognized as \textit{10 writer little} by ASR. The most probable intention returned by HTMM for this utterance is \( i_3 : \text{turn right little} \) with 0.99 probability. This is because HTMM considers Markovian property for deriving intentions, cf. Section 4.2.1.
... 

$u_1$: turn right 
$ar{u}_1$: [turn right]

$Pr_1$

\[
i_1: 7.1e-9 \quad i_2: 9.6e-10 \quad i_3: 0.6
\]

$Pr_2$

\[
i_4: 0.2 \quad i_5: 2.6e-8 \quad i_6: 2.2e-5
\]

$Pr_3$

\[
i_4: 0.0 \quad i_5: 1.3e-7 \quad i_6: 5.8e-8
\]

$Pr_4$

\[
i_7: 0.1 \quad i_8: 6.3e-7 \quad i_9: 1.6e-8 \quad i_{10}: 2.4e-6 \quad i_{11}: 5.2e-9
\]

$a_1$: TURN RIGHT

$u_2$: turn right a little 
$ar{u}_2$: [10 writer little]

$Pr_2$

\[
i_1: 0.0 \quad i_2: 0.0 \quad i_3: 0.9
\]

$Pr_3$

\[
i_4: 0.0 \quad i_5: 1.3e-7 \quad i_6: 5.8e-8
\]

$Pr_4$

\[
i_7: 8.8e-8 \quad i_8: 1.2e-6 \quad i_9: 5.9e-5 \quad i_{10}: 8.8e-5 \quad i_{11}: 1.1e-7
\]

$a_2$: PLEASE REPEAT YOUR COMMAND

$u_3$: turn right a little 
$ar{u}_3$: [turn right to lead a]

$Pr_3$

\[
i_4: 0.0 \quad i_5: 2.7e-08 \quad i_6: 2.0e-07
\]

$Pr_4$

\[
i_7: 0.0 \quad i_8: 3.9e-9 \quad i_9: 1.9e-10 \quad i_{10}: 4.4e-08 \quad i_{11}: 1.7e-11
\]

$a_3$: TURN RIGHT A LITTLE

$u_4$: stop 
$ar{u}_4$: [stop]

$Pr_4$

\[
i_1: 3.2e-5 \quad i_2: 4.8e-6 \quad i_3: 0.0
\]

$Pr_4$

\[
i_4: 0.0 \quad i_5: 0.0 \quad i_6: 7.8e-6
\]

$Pr_4$

\[
i_7: 0.0 \quad i_8: 0.0 \quad i_9: 0.0 \quad i_{10}: 0.0 \quad i_{11}: 0.9
\]

$a_4$: STOP

...
Consequently, in the second turn the intention $i_3$ gets high probability since in the first turn the user intention is $i_3$ with high probability.

Before we learn a complete dialogue POMDP, first we learned a dialogue MDP using the SmartWheeler dialogues. We used the learned intentions, $i_1, \ldots, i_{11}$, as the states of the MDP. The learned states are presented in Table 6.5. Note that for the intention $i_7$, we used it as the state for the command *turn degree right* as in the intention $i_7$ the word *right* occurs with slightly higher probability than the word *left*.

| $s_1$ | move-forward-little |
| $s_2$ | move-backward-little |
| $s_3$ | turn-right-little |
| $s_4$ | turn-left-little |
| $s_5$ | follow-left-wall |
| $s_6$ | follow-right-wall |
| $s_7$ | turn-degree-right |
| $s_8$ | go-door |
| $s_9$ | set-speed |
| $s_{10}$ | follow-person |
| $s_{11}$ | stop |

**Table 6.5:** The SmartWheeler learned states.

Then, we learned the transition model, i.e., the smoothed maximum likelihood transition method, introduced in Section 4.3. Note that the dialogue MDP here is in fact an intention dialogue MDP in the same way defined in Section 4.4. That is, we used a deterministic intention observation model for the dialogue MDP, which considers the observed intention as its current state during the dialogue interaction.

### 6.2.1 Observation model learning

Built off the learned dialogue MDP, we developed two dialogue POMDPs by learning the two observation sets and their subsequent observation models: keyword model and intention model, proposed in Section 4.4. From these models, we then developed the keyword dialogue POMDP and the intention dialogue POMDP for SmartWheeler. As mentioned in Section 4.5.2, here we show the two observation sets for SmartWheeler and then compare the intention POMDP performance to the keyword POMDP performance.

The keyword observation model for each state uses a keyword that best represents the state. We use the *1-top* word of each state, shown in Table 6.3, as observations (the highlighted words). That is, the observations are:

forward, backward, right, left, turn, go, for, top, stop.
Note that states $s_3$ and $s_6$ share the same keyword observation, i.e. right. Also, states $s_4$ and $s_5$ share the same keyword observation, i.e., left.

For the intention model, each state itself is the observation. Then, the set of observations is equivalent to the set of intentions. For SmartWheeler the intention observations are:

$$i_1o, i_2o, i_3o, i_4o, i_5o, i_6o, i_7o, i_8o, i_9o, i_{10}o, i_{11}o.$$ respectively for the states:

$$s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}.$$ Table 6.6 shows the sample dialogue from SmartWheeler after learning the two observation sets. In this table, line $o_1$ is the observation for the recognized utterance by ASR, $\tilde{u}_1$. If the keyword observation model is used the observation will be right, however, if intention observation model is used then the observation will be the one inside parenthesis, i.e., $i_3o$. In fact, $i_3o$ is an observation with high probability for the state $s_3$, and with low probability for the rest of states.

Note that in $o_2$ for the case of keyword observation, the observation is confusedObservation. This is because for the keyword model, none of the keyword observations occurs in the recognized utterance $\tilde{u}_2$. However, the intention observation interestingly becomes $i_3o$ which is the same as the intention observation in $o_1$.

\begin{table}[h]
\centering
\begin{tabular}{|l|}
\hline
$\ldots$  \\
$u_1$ : turn right  \\
$\tilde{u}_1$ : [turn right]  \\
$o_1$ : right $(i_3o)$  \\
\hline
$u_2$ : turn right a little  \\
$\tilde{u}_2$ : [10 writer little]  \\
$o_2$ : confusedObservation $(i_3o)$  \\
\hline
$u_3$ : turn right a little  \\
$\tilde{u}_3$ : [turn right to lead a]  \\
$o_3$ : right $(i_3o)$  \\
\hline
$u_4$ : stop  \\
$\tilde{u}_4$ : [stop]  \\
$o_4$ : stop $(i_{11}o)$  \\
$\ldots$  \\
\hline
\end{tabular}
\caption{A sample from the results of applying the two observation models on the SmartWheeler dialogues.}
\end{table}
6.2.2 Comparison of the intention POMDP to the keyword POMDP

As mentioned in Section 4.5.2, we compared the keyword POMDP to the intention POMDP. Recall from the previous section that in the keyword POMDP, the observation set is the set of learned keywords and the observation model is the learned keyword observation model. In the intention POMDP, however, the observation set is the set of learned intentions and the observation model is the learned intention observation model. The learned keyword and intention POMDPs are then compared based on their policies. To do so, we assumed a reward model for the two dialogue POMDPs and compared the optimal policies of the two POMDPs, based on their accumulated mean rewards in simulation runs.

Similar to the previous work of Png and Pineau [2011], we considered reward of +1 for the SmartWheeler performing the right action at each state, and 0 otherwise. Moreover, for the general query, PLEASE REPEAT YOUR COMMAND, the reward is considered as +0.4 for each state where this query occurs. The intuition for this reward is that in each state it is best to perform the right action of the state, and it is better to perform a general query action than to perform any other wrong action in the state. That is the reason for defining the +0.4 reward for the query action (0 < +0.4 < 1). This reward model is represented in Table 6.9 (top), which is also used as the expert reward model in the IRL experiments in Section 6.3.

The dialogue POMDP models consist of 11 states, 12 actions and 10 observations if the keyword observation model is used (9 keywords and the confusedObservation). Otherwise, there are 11 observations for the intention observation model. We solved our POMDP models, using ZMDP software available online at: http://www.cs.cmu.edu/~trey/zmdp/. We set a uniform distribution on states, and set the discount factor to 0.90.

Similar to Section 4.5.2, we evaluated our learned observation models based on accumulated mean rewards. This is because the reward model is the same for the intention POMDP and keyword POMDP. Then, the learned policy of each model can reflect the quality of the learned observation model.

We used the default simulation in ZMDP software which simulates the environment by randomly sampling observations and uses the provided observation and transition models. Note that since the transition model is the same for the intention POMDP and keyword POMDP, the accumulated reward by policy of each model can demonstrate the quality of the observation model.

Table 6.7 shows the comparison of the two models based on 1000 simulation runs. The
table shows that the intention POMDP accumulates strongly higher mean reward than the keyword POMDP based on 1000 simulation runs by ZMDP software. In Table 6.7, \textit{Conf95Min} and \textit{Conf95Max} are respectively the minimum 95\% confidence and the maximum 95\% confidence of the accumulated mean reward. This means that with approximately 95\% confidence the accumulated mean reward occurs inside the interval formed by \textit{Conf95Min} and \textit{Conf95Max}.

As such, we perform the POMDP-IRL experiments for learning the reward model from SmartWheeler dialogues on the learned intention POMDP. Similarly, we perform the MDP-IRL experiments on the learned intention MDP, i.e., the intention POMDP with the deterministic observation model.

<table>
<thead>
<tr>
<th></th>
<th>Mean Reward</th>
<th>\textit{Conf95Min}</th>
<th>\textit{Conf95Max}</th>
</tr>
</thead>
<tbody>
<tr>
<td>intention POMDP</td>
<td>8.914</td>
<td>8.904</td>
<td>8.922</td>
</tr>
<tr>
<td>keyword POMDP</td>
<td>4.784</td>
<td>4.767</td>
<td>4.802</td>
</tr>
</tbody>
</table>

\textbf{Table 6.7}: The performance of the intention POMDP vs. the keyword POMDP, learned from the SmartWheeler dialogues.

### 6.3 Reward model learning for SmartWheeler

In this section, we experiment the MDP-IRL algorithm, introduced in Section 5.2 and the POMDP-IRL-BT algorithm, proposed in Section 5.3.1. As mentioned in Section 5.1, the IRL experiments are designed to verify if the introduced IRL methods are able to learn a reward model for the expert policy, where the expert policy is represented as a (PO)MDP policy. That is, the expert policy is the optimal policy of the (PO)MDP with a known model. Thus, similar to section 5.6, we assumed an expert reward model \( R^\pi_E \) and used the (PO)MDP model to find the expert policy \( \pi_E \). The learned expert policy was used to sample \( B \) expert trajectories to be used in the IRL algorithms.

Based on the experiments in the previous section, we selected the intention MDP/POMDP to be used as the underlying MDP/POMDP framework. The intention POMDP consists of 11 states, 24 actions, 11 intention observations, and the learned transition and observation models. The initial belief, \( b_0 \), is set to the uniform belief. The intention MDP is similar to the intention POMDP, but the observation model is deterministic.

#### 6.3.1 Choice of features

Recall from the previous chapter that IRL needs features to represent the reward model. We propose \textit{keyword} features for applying IRL on the learned dialogue MDP/POMDP.
from SmartWheeler. The keyword features are SmartWheeler keywords, i.e., 1-top words for each user intention from Table 6.3. There are nine learned keywords:

\[ \text{forward, backward, right, left, turn, go, for, top, stop.} \]

The keyword features for each state of SmartWheeler dialogue POMDP are represented in a vector, as shown in Table 6.8. The figure shows that states \( s_3 \) (\text{turn-right-little}) and \( s_6 \) (\text{follow-right-wall}) share the same features, i.e., \text{right}. Moreover, states \( s_4 \) (\text{turn-left-little}) and \( s_5 \) (\text{follow-left-wall}) share the same feature, i.e., \text{left}. In our experiments, we used \text{keyword-action-wise} features. Such features include the indicator functions for each pair of state-keyword and action. Thus, the feature size for SmartWheeler equals \( 216 = 9 \times 24 \) (9 keywords and 24 actions).

Note that the choice of features is application dependent. The reason for using keywords as state features is that in the intention-based dialogue applications the states are the dialogue intentions, where each intention is described as a vector of k-top words from the domain dialogues. Therefore, the keyword features are relevant features for the states.

Note also that although the keyword features are similar to the keyword observations proposed for POMDP observations in Section 4.4, there is no explicit learned model for their dynamics such as the keyword observation model proposed in Section 4.4. In particular, for MDPs there is no observation model, however the keyword features are used in MDP-IRL for the reward model representation.

<table>
<thead>
<tr>
<th></th>
<th>forward</th>
<th>backward</th>
<th>right</th>
<th>left</th>
<th>turn</th>
<th>go</th>
<th>for</th>
<th>top</th>
<th>stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_1 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_4 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_5 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_6 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_7 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_8 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_9 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_{10} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_{11} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\text{Table 6.8: Keyword features for the SmartWheeler dialogues.}
6.3.2 MDP-IRL learned rewards

In this section, we show the learned reward model by the MDP-IRL algorithm for the expert policy, where similar to previous works [Ng and Russell, 2000; Choi and Kim, 2011], the expert policy is a MDP policy (cf. Section 5.1). To do so, we assumed an expert reward model for the learned intention MDP from SmartWheeler. We then solved the model to find the (near) optimal policy which is used as the expert policy.

Similar to the previous section, we assumed the reward model used in Png and Pineau [2011]. Table 6.9 (top) shows the expert reward model. That is, we considered +1 reward for performing the right action at each state, and 0 otherwise. Moreover, for the general query PLEASE REPEAT YOUR COMMAND in every state the reward is considered as +0.4. We then solved the intention MDP model with the assumed expert reward to find the optimal policy, i.e., the expert policy. The expert policy for each of the MDP state is represented in Table 6.10. Interestingly, the expert policy suggests performing the right action of each state.

We then applied the MDP-IRL algorithm on SmartWheeler dialogue MDP described above using the introduced keyword features in Table 6.8. The algorithm was able to learn a reward model in which the policy equals the expert policy for all states, (the expert policy shown in Table 6.10). Table 6.9 (bottom) shows the learned reward model. Comparing the assumed expert reward model in Table 6.9 (top) to the learned reward model in Table 6.9 (bottom), we observe that the rewards in the two tables are different, however, the policy of the learned reward model is exactly the same as expert policy (shown in Table 6.10). The difference of the two reward models with the same policy is since IRL is an ill-posed problem, as mentioned in Section 5.1.

6.3.3 POMDP-IRL-BT evaluation

In this section, we show our experiments on the POMDP-IRL-BT algorithm on the intention dialogue POMDP learned from SmartWheeler. As mentioned earlier, to evaluate the IRL algorithms, we consider that expert policy is a POMDP policy using an assumed reward model. Similar to previous section, we assumed that the expert reward model is the one represented in Table 6.9 (top). For the choice of features, we also used the keyword features shown in Table 6.8.

Similar to the experiments in Section 5.6, we performed two fold cross validation experiments by generating 10 expert trajectories. The expert trajectories are truncated after 20 steps, since there is no terminal state here. We then used the Perseus software with the same setting as described in Section 5.6. That is, we set the solver to use 10,000 random samples for solving the optimal policy of each candidate reward. The
Chapter 6. Application on healthcare dialogue management

Assumed expert reward model

<table>
<thead>
<tr>
<th></th>
<th>a₁</th>
<th>a₂</th>
<th>a₃</th>
<th>a₄</th>
<th>a₅</th>
<th>a₆</th>
<th>a₇</th>
<th>a₈</th>
<th>a₉</th>
<th>a₁₀</th>
<th>a₁₁</th>
<th>a₁₂</th>
<th>...</th>
<th>REPEAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0.4</td>
</tr>
<tr>
<td>s₂</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0.4</td>
</tr>
<tr>
<td>s₃</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0.4</td>
</tr>
<tr>
<td>s₄</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0.4</td>
</tr>
<tr>
<td>s₅</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0.4</td>
</tr>
<tr>
<td>s₆</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0.4</td>
</tr>
<tr>
<td>s₇</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0.4</td>
</tr>
<tr>
<td>s₈</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0.4</td>
</tr>
<tr>
<td>s₉</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Learned reward model by MDP-IRL

<table>
<thead>
<tr>
<th></th>
<th>a₁</th>
<th>a₂</th>
<th>a₃</th>
<th>a₄</th>
<th>a₅</th>
<th>a₆</th>
<th>a₇</th>
<th>a₈</th>
<th>a₉</th>
<th>a₁₀</th>
<th>a₁₁</th>
<th>a₁₂</th>
<th>...</th>
<th>REPEAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>s₂</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>s₃</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>s₄</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>s₅</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>s₆</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>s₇</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>s₈</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>s₉</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>s₁₀</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>s₁₁</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.9: Top: The assumed expert reward model for the dialogue MDP/POMDP learned from SmartWheeler dialogues. Bottom: The learned reward model for the learned dialogue MDP from SmartWheeler dialogues using keyword features.

other parameter is max-time for execution of the algorithm, which is set to 1000.

Based on the specification above, we performed POMDP-IRL-BT on SmartWheeler expert trajectory for training. The experimental results showed that the policy of the learned reward was the same as the expert policy for 194 beliefs inside the testing trajectory out of the 200 beliefs, i.e., 97% matched actions. For all the 6 errors, the expert action was TURN RIGHT LITTLE, i.e., the right action for the state turn-right-little, while the action of the learned reward suggested FOLLOW RIGHT WALL. However, this error did not happen in all the cases which the expert action was TURN RIGHT LITTLE in the testing trajectory.

Afterwards, we used state-action-wise features as defined in Section 5.6. Such features include an indicator function for each state-action pair. In SmartWheeler, there are
Table 6.10: The policy of the learned dialogue MDP from SmartWheeler dialogues with the assumed expert reward model.

<table>
<thead>
<tr>
<th>state</th>
<th>state description</th>
<th>expert action</th>
<th>expert action description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>move-forward-little</td>
<td>a1</td>
<td>DRIVE FORWARD A LITTLE</td>
</tr>
<tr>
<td>s2</td>
<td>move-backward-little</td>
<td>a2</td>
<td>DRIVE BACKWARD A LITTLE</td>
</tr>
<tr>
<td>s3</td>
<td>turn-right-little</td>
<td>a3</td>
<td>TURN RIGHT A LITTLE</td>
</tr>
<tr>
<td>s4</td>
<td>turn-left-little</td>
<td>a4</td>
<td>TURN LEFT A LITTLE</td>
</tr>
<tr>
<td>s5</td>
<td>follow-left-wall</td>
<td>a5</td>
<td>FOLLOW THE LEFT WALL</td>
</tr>
<tr>
<td>s6</td>
<td>follow-right-wall</td>
<td>a6</td>
<td>FOLLOW THE RIGHT WALL</td>
</tr>
<tr>
<td>s7</td>
<td>turn-degree-right</td>
<td>a7</td>
<td>TURN RIGHT DEGREES</td>
</tr>
<tr>
<td>s8</td>
<td>go-door</td>
<td>a8</td>
<td>GO THROUGH THE DOOR</td>
</tr>
<tr>
<td>s9</td>
<td>set-speed</td>
<td>a9</td>
<td>SET SPEED TO MEDIUM</td>
</tr>
<tr>
<td>s10</td>
<td>follow-wall</td>
<td>a10</td>
<td>FOLLOW THE WALL</td>
</tr>
<tr>
<td>s11</td>
<td>stop</td>
<td>a11</td>
<td>STOP</td>
</tr>
</tbody>
</table>

11 states and 24 actions, then the size of state-action-wise features equals $264 = 11 \times 24$. This is a slight increase compared to the size of keyword features, i.e., 216. We observed that in our experiment the learned policy is exactly the same as the expert policy for the 200 beliefs inside the testing trajectory using state-action-wise features, i.e., 100% matched with the expert policy. In words, POMDP-IRL-BT was able to learn a reward model for the expert policy using the learned dialogue POMDP from SmartWheeler dialogues. In the following section, we compare POMDP-IRL-BT to POMDP-IRL-MC introduced in Section 5.5, in which the policy values are estimated using the Monte Carlo estimator rather than by approximating the belief transitions.

6.3.4 Comparison of POMDP-IRL-BT to POMDP-IRL-MC

In Section 5.4, we saw that Choi and Kim [2011] proposed IRL algorithms in POMDP framework by assuming policies in the form of an FSC and thus using PBPI (point-based policy iteration) [Ji et al., 2007], as POMDP solver. In their algorithm, they used Monte Carlo estimator to estimate the value of expert policy whereas we used an estimated belief transition model for the expert beliefs to be able to use bellman equation for approximating the expert policy values as well as candidate policy values. As stated in Section 5.5, we also implemented the Monte Carlo estimator (Equation (5.19)) for the estimation of policy values in Line 7 in Algorithm 8, and used the Perseus software [Spaan and Vlassis, 2005] as the POMDP solver. This new algorithm is called POMDP-IRL-MC. We compared POMDP-IRL-BT to POMDP-IRL-MC. The purpose of such experiments was to compare the belief transition estimation to the Monte Carlo estimation.
We compared the two algorithms, POMDP-IRL-BT and POMDP-IRL-MC, based on the following criteria:

1. Percentage of the learned actions that matches to the expert actions.

2. Value of learned policy with respect to the value of expert policy.

3. CPU time spent by the algorithm as the number of expert trajectories (training data) increases.

Criteria 1 and 2 are used to evaluate the quality of the learned reward model for the expert. As in the previous experiment, the higher the matched actions, the better the learned reward model is. Similarly, criterion 2 compares the value of the learned reward model with the value of expert reward model. The higher the value of the learned policy, the better the learned reward model is. The results for these criteria is based on two fold cross validation using 400 expert trajectories, i.e., each fold contains of 200 expert trajectories.

Note that the value of learned policy (in criterion 2) is the sampled value of the policy. This was done by running the policy starting from a uniform belief to the maximum \( maxT = 20 \) time step or until it reaches the terminal state. The sampled values are averaged over 100 runs, and are calculated using:

\[
\hat{V}^\pi (b) = \left[ \sum_{t=0}^{maxT} \gamma^t R(b_t, \pi(b_t)) \| \pi, b_0 = b \right]
\]

Finally, criterion 3 evaluates the CPU time spent by the algorithm as the number of expert trajectories increases. This is to verify which of the two algorithms, POMDP-IRL-BT and POMDP-IRL-MC, requires more computation time. Below, we report on our experiments on SmartWheeler domain based on the above mentioned criteria.

### 6.3.4.1 Evaluation of the quality of the learned rewards

First, we evaluated POMDP-IRL-BT and POMDP-IRL-MC using keyword features based on criteria 1 and 2. The results are shown in Figure 6.2 (top) and Figure 6.2 (bottom). The two figures show consistent results in which the performance of POMDP-IRL-BT and POMDP-IRL-MC are comparable.

Figure 6.2 (top) shows percentage of the matched actions to those of expert, as the number of iterations increases (the first criteria). The figure demonstrates that after around 15 iterations the learned actions for 95% of testing trajectories matches to actions suggested by the expert policy, in both the POMDP-IRL-BT and POMDP-IRL-MC algorithms. The figure also shows that after iteration 15, percentage of the matched
Figure 6.2: Comparison of the POMDP-IRL algorithms using keyword features on the learned dialogue POMDP from SmartWheeler. Top: percentage of matched actions. Bottom: sampled value of the learned policy.
actions fluctuates slightly as the number of iterations increases, however percentage remains above 90%.

Moreover, Figure 6.2 (bottom) plots the value of the learned policy (the sampled value) as the number of iterations increases (criterion 2). Similar to Figure 6.2 (top), we observe that for both POMDP-IRL-BT and POMDP-IRL-MC after iteration 15 the learned policy value becomes close to the expert policy value. Moreover, though the learned policy values fluctuate slightly, it remains close to the expert policy value after iteration 15.

The reason for these fluctuations is the choice of features. In the experiments reported above we used the automatically learned keyword features for our POMDP-IRL experiments. In Table 6.8, we saw that the states 3 and 6 share the same feature right. Similarly, the states 4 and 5 share the same feature left. Although this kind of feature sharing can reduce the size of features, it can lead to learning wrong actions for the sharing states.

Therefore, we performed similar experiments on SmartWheeler but this time using state-action features. These features include the indicator functions for each pair of state and action. Thus, the feature size for SmartWheeler equals $11 \times 24 = 264$, which is a slight increase compared to the size of keyword features, i.e., 216. Similar to the keyword features, we evaluated state-action features on SmartWheeler based on criteria 1 and 2. The results are shown in Figure 6.3 (top) and Figure 6.3 (bottom).

Figure 6.3 (top) and Figure 6.3 (bottom) show consistent results in which the performance of POMDP-IRL-BT reaches to expert performance. Figure 6.3 (top) shows percentage of the matched actions between the learned and expert policies, as the number of iterations increases. The figure shows that this percentage reaches to 100% in POMDP-IRL-BT, while it reaches to 97% in POMDP-IRL-MC.

Moreover, Figure 6.3 (bottom) plots the value of the learned policy as the number of iterations increases. We observe that the learned value equals the value of expert policy in POMDP-IRL-BT (at iteration 13), while in POMDP-IRL-MC it only gets close to the value of expert policy (at iteration 17). Furthermore, Figure 6.3 (top) and Figure 6.3 (bottom) show that using state-action features, POMDP-IRL-BT reaches its optimal performance (equal to the expert performance) slightly earlier than POMDP-IRL-MC (at iteration 13 and iteration 17, respectively).

6.3.4.2 Evaluation of the spent CPU time

Figure 6.4 demonstrates the spent time by POMDP-IRL-BT and POMDP-IRL-MC as the number of expert trajectories (training data) increases. The results show that by
Figure 6.3: Comparison of the POMDP-IRL algorithms using state-action-wise features on the learned dialogue POMDP from SmartWheeler. Top: percentage of matched actions. Bottom: sampled value of learned policy.
increasing the number of expert trajectories, POMDP-IRL-BT requires considerably more time than POMDP-IRL-MC. Note that the figure plots the spent time by the number of trajectories in the log base. This increase is due to increase of the size of belief transition matrix, Equation (5.12), as the number of expert trajectories increases. In other words, the belief transition matrix requires much more time to be constructed as the number of beliefs in expert trajectories increases. Also, note that this matrix is constructed for each candidate policy, which in turn increases the CPU time.

In sum, our experimental results showed that using state-action features, the POMDP-IRL-BT is able to learn a reward model in which the policy matches the expert policy for 100% of beliefs in the testing trajectories, while POMDP-IRL-MC learned a reward model in which the policy matched the expert policy for only 97% of beliefs in testing trajectories. However, POMDP-IRL-MC does scale substantially better than POMDP-IRL-BT. In the case of large number of expert trajectories, POMDP-IRL-BT can still be useful. For instance, we can use all expert trajectories to estimate the transition and observation models, but, select part of the expert trajectories to learn the reward model.
6.4 Conclusions

In this chapter, we applied the proposed methods in this thesis on a healthcare dialogue management. We used the dialogues collected by an intelligent wheelchair called SmartWheeler for learning the model components of the dialogue POMDP. To do so, we first learned the user intentions that occurred in the SmartWheeler dialogues and used them as states of the dialogue POMDP. Then, we used the learned states and the extracted SmartWheeler actions to learn the maximum likelihood transition model.

For the observation model of SmartWheeler dialogue POMDP, we learned both the intention and keyword observation models. We observed that the intention POMDP, i.e., the POMDP using the intention observation model, performed significantly better than the keyword POMDP.

We then introduced the automatically learned keyword features and applied the MDP-IRL algorithm, introduced in the previous chapter, on the learned intention MDP from SmartWheeler. The algorithm learned a reward model whose policy completely matched to the expert policy using the keyword-action-wise features. Furthermore, we evaluated our proposed POMDP-IRL-BT algorithm on the learned intention POMDP from SmartWheeler. We observed that POMDP-IRL-BT is able to learn a reward model that accounts for the expert policy using keyword-action-wise and state-action-wise features.

Finally, we compared the POMDP-IRL-BT algorithm to the POMDP-IRL-MC algorithm which uses Monte Carlo estimation in the place of belief transition estimation. Our experiments showed that the both algorithms are able to learn a reward model that accounts for the expert policy using keyword-action-wise and state-action-wise features. Furthermore, our experimental results showed that POMDP-IRL-BT slightly outperforms the POMDP-IRL-MC algorithm, however, the POMDP-IRL-MC does scale better than POMDP-IRL-BT.

Overall, the experiments on SmartWheeler dialogues showed that the proposed methods are able to learn the dialogue POMDP model components from real dialogues. In the following section, we summarize the thesis and address multiple avenues for future research of dialogue POMDP model learning.
Chapter 7

Conclusions and future work

7.1 Thesis summary

Spoken dialogue systems (SDSs) are the systems that help the human user to accomplish a task using the spoken language. Dialogue management is a difficult problem since automatic speech recognition (ASR) and natural language understanding (NLU) make errors which are the sources of uncertainty in SDSs. Moreover, the human user behavior is not completely predictable. The users may change their intentions during the dialogue, which makes the SDS environment stochastic. Furthermore, the users may express an intention in several ways which makes dialogue management more challenging.

In this context, partially observable Markov decision process (POMDP) framework has been used to model the dialogue management of spoken dialogue systems. The POMDP framework can deal with both the uncertainty and stochasticity in the environment in a principled way. Furthermore, the POMDP framework has shown better performance compared to other frameworks, such as Markov decision processes (MDPs). This is particularly the case in the noisy environments, which is often the case in spoken dialogue systems.

However, POMDPs and their application on spoken dialogue systems involve many challenges. In particular, we were mostly interested in learning the dialogue POMDP model components from unannotated and noisy dialogues. In this context, there are a large number of unannotated dialogues available which can be used for learning dialogue POMDP model components. In addition, learning the dialogue POMDP model components from data is particularly significant since the learned dialogue POMDP model directly affects the POMDP policy. Furthermore, learning proper dialogue POMDP model components from real data could be highly beneficial since there is a rich literature on model-based POMDP solving that can be used once the dialogue POMDP
model components are learned. In words, if we are able to learn a realistic dialogue POMDP from data, then we can make use of available POMDP solvers for learning the POMDP policy.

In this thesis, we proposed methods for learning dialogue POMDP model components from unannotated dialogues for intention-based dialogue domains in which the user intention is the dialogue state. We demonstrated the big picture of our approach in a descriptive algorithm (Algorithm 1). Our POMDP model learning approach started by learning the dialogue POMDP states. The learned states were then used for learning the transition model followed by the dialogue POMDP observations and observation model. Building off these learned dialogue POMDP model components, we proposed two POMDP-IRL algorithms for learning the reward model.

For the dialogue states, we learned the possible user intentions that appeared in the user dialogues using a unsupervised topic modeling method. In this way, we were able to learn the user intentions from unannotated dialogues and used them as the dialogue POMDP states. To do so, we used HTMM (hidden topic Markov model) which is a variation of latent Dirichlet allocation (LDA) that considers the Markovian property between dialogues. Using the learned intentions as the dialogue states, and the set of actions, extracted from the dialogues, we learned a maximum likelihood transition model for the dialogue POMDP. We then proposed two observation models: the keyword model and the intention model. The keyword model used only the learned keywords, from the topic modeling approach, as the set of observations. The intention model, however, used the set of intentions as the set of observations. As the two models include a small number of observations, solving the POMDP model becomes tractable.

Furthermore, we introduced trajectory-based inverse reinforcement learning (IRL) for learning the reward model in the (PO)MDP framework using expert trajectories. In this context, we introduced the MDP-IRL algorithm, the basic IRL algorithm in the MDP framework. We then proposed two POMDP-IRL algorithms: POMDP-IRL-BT and PB-POMDP-IRL. The POMDP-IRL-BT algorithm is similar to the MDP-IRL. However, POMDP-IRL-BT uses belief states rather states, and approximates a belief transition model, which is similar to the state transition model in MDPs. On the other hand, PB-POMDP-IRL is a point-based POMDP-IRL algorithm that approximates the value of the new beliefs, which occurs in the computation of the policy values, using a linear approximation of expert beliefs. The two algorithms are able to learn a reward model that accounts for expert policy. However, our experimental results showed that POMDP-IRL-BT outperforms PB-POMDP-IRL since the policy of learned reward model by the former algorithm matched with more expert actions.

We then applied the proposed methods in this thesis to learn a dialogue POMDP from dialogues collected in a healthcare domain. That is, we used the dialogues collected by
SmartWheeler, an intelligent wheelchair for handicapped people. We were able to learn 11 user intentions, which were considered as states of the dialogue POMDP. Based on the learned intentions and the SmartWheeler actions, we then learned the maximum likelihood transition model. We then learned the two observation sets and their subsequent observation models: the keyword and intention models. Our experimental results showed that the intention model outperforms the keyword model-based on accumulated mean rewards in simulation runs. We thus used the learned intention POMDP for the rest of experiments, i.e., for IRL evaluations.

To perform the IRL experiments, we introduced the automatically learned keyword features. We then applied the MDP-IRL algorithm, on the learned intention MDP from SmartWheeler. The algorithm learned a reward model whose policy completely matched to the expert policy using the keyword-action-wise features. Furthermore, we evaluated the POMDP-IRL-BT algorithm on the learned intention POMDP from SmartWheeler. We observed that POMDP-IRL-BT is able to learn a reward model that accounts for the expert policy using keyword-action-wise features.

Finally, we compared the POMDP-IRL-BT algorithm that uses belief transition estimation to the POMDP-IRL-MC algorithm that uses Monte Carlo estimation. Our experimental results showed that the both algorithms are able to learn a reward model that accounts for the expert policy. Furthermore, the results showed that POMDP-IRL-BT slightly outperforms the POMDP-IRL-MC algorithm based on matched actions to the expert actions as well as the learned policy values. On the other hand, the POMDP-IRL-MC algorithm does scale better than the POMDP-IRL-BT algorithm.

### 7.2 Future work

This thesis can be extended in several directions. In particular, we used HTMM to learn the dialogue POMDP intentions, mainly because HTMM considers the Markovian property inside dialogues and it is computationally efficient. One direction for future work can be application of other topic modeling approaches such as the LDA [Blei et al., 2003]. A survey of topic modeling methods can be found in Blei [2011]; Daud et al. [2010]. Moreover, for the transition model we used the add-one smoothed transition model due to its simplicity and sufficiency for the purpose of our experiments. However, there are many other smoothing approaches in the literature which can be tested and compared to the introduced add-one smoothed transition model. For a comprehensive background on smoothing techniques the reader is refereed to Manning and Schütze [1999]; Jurafsky and Martin [2009].

We proposed two sets of observations and their subsequent observation models. The pro-
posed learned observation models could be further extended and enhanced for instance by merging the keyword observations and intention observations, considering multiple top keywords of each state rather than considering only one keyword. Furthermore, other methods could be used for learning the observation model such as Bayesian-based methods [Atrash and Pineau, 2010; Doshi and Roy, 2008; Png and Pineau, 2011]. In particular, Png and Pineau [2011] proposed an online Bayesian approach for updating the observation model which can be extended for learning the observation model of dialogue POMDPs from SmartWheeler dialogues.

In this thesis, we introduced the basic MDP-IRL algorithm of Ng and Russell [2000], and extended it for POMDPs. However, there are a vast number of IRL algorithms in the MDP framework [Abbeel and Ng, 2004; Ramachandran and Amir, 2007; Neu and Szepesvári, 2007; Syed and Schapire, 2008; Ziebart et al., 2008; Boularias et al., 2011]. The MDP-IRL algorithms can potentially be extended to POMDPs [Kim et al., 2011]. In particular, Kim et al. [2011] extended the MDP-IRL algorithm of Abbeel and Ng [2004], which is called max-margin between feature expectations (MMFE), to a finite state controller (FSC) based POMDP-IRL algorithm. The authors showed that the extension of MMFE for POMDPs performs pretty well based on experiments on several POMDP benchmarks. The MMFE POMDP algorithm of [Kim et al., 2011] also could be extended as a point-based POMDP-IRL algorithm in order to take advantage of the computational efficiency of point-based POMDP solvers such as Perseus.

Furthermore, the IRL algorithms requires (dialogue) features for representing the reward model. A relevant reward model to the dialogue system and users can be only learned by studying and extracting relevant features from the dialogue domain. Future research should be devoted on automatic methods for learning the relevant and proper features that are suitable for reward representation and reward model learning. We also observed that POMDP-IRL-BT algorithm does not scale as the number of trajectories increase. Although, the scalability may not be a great issue as the algorithm can learn the reward model of the expert using a small number of trajectories, another future avenue of research can be enhancing the scalability of the POMDP-IRL-BT algorithm.

Ultimately, in this thesis, we considered intention-based dialogue POMDPs particularly because they can have large applications, for instance in spoken web search. Our dialogue POMDPs currently deal with small set of intentions; they can however be extended to larger domains for instance by considering the domain’s hierarchy, and considering a dialogue POMDP for each level of the hierarchy. Furthermore, the developed techniques in other dialogue domains can be incorporated for intention-based dialogue POMDPs, such as factored-based transition and observation model [Williams, 2006].
Appendix A

IRL

This appendix includes two sections including materials related to IRL, presented in Chapter 5. The materials in this appendix have been developed during the author’s internship at AT&T research labs in summer 2010 and the author’s collaboration with AT&T research labs during 2011.

Section A.1 demonstrates an experiment showing that IRL is an ill-posed problem, introduced in Section 5.1. Section A.2 presents a model-free trajectory-based MDP-IRL algorithm, called LSPI-IRL, in which the candidate policies (optimal policy of candidate rewards) are estimated using the LSPI (least-squares policy iteration) algorithm [Lagoudakis and Parr, 2003]. We then show the performance of LSPI-IRL. We show that this algorithm is able to learn a reward model that accounts for expert policy using state-action-wise features. We then show that the LSPI-IRL performance decreases as the expressive power of the used features decreases.

A.1 IRL, an ill-posed problem

In Section 5.1, we mentioned that IRL is an ill-posed problem since there is a set of reward models that make the expert policy optimal. In this section, we demonstrate an experiment showing that there is a wide space in which the reward models can make the expert policy optimal.

The experiments in this appendix are performed on a MDP defined for the 3-slot problem in which the machine should obtain the values for three assumed slots. Each slot can take four ASR confidence score values:

\textit{empty, low, medium, and high}. 
The machine’s actions are:

- **Ask-slot-i**, **Confirm-slot-i**, **Ask-all** slots, and **Submit**.

As such, for the 3-slot problem, there are $64 = 4^3$ states (3 slots and 4 values). And, there are 8 actions: 3 **Ask-slot-i** actions (one for each slot), 3 **Confirm-slot-i** actions (one for each slot), the **Ask-all**, and the **Submit** actions.

We assumed that the reward model for the 3-slot problem is defined as:

$$R(s, a) = \begin{cases} 
    w_1 f_1 + w_2 f_2 & \text{if } a = \text{Submit} \\
    -1 & \text{Otherwise} 
\end{cases}$$  \hspace{1cm} (A.1)

in which the feature weights are set as: $w_1 = +20$ and $w_2 = -10$, for the defined features as follows:

- $f_1$: the probability of successful task completion, i.e., probability of executing the **Submit** action correctly, denoted by $f_1 = p(C)$,
- $f_2$: the probability of unsuccessful task completion, denoted by $f_2 = 1 - p(C)$.

More specifically, for the 3-slot problem, the probability of executing the **Submit** action correctly is defined as:

$$p(C) = p(C \text{ slot 1}) \ast p(C \text{ slot 2}) \ast p(C \text{ slot 3})$$

in which

$$p(C \text{ slot } i) = \begin{cases} 
    0 & \text{if the value of slot } i \text{ is empty} \\
    0.3 & \text{if the value of slot } i \text{ is low} \\
    0.5 & \text{if the value of slot } i \text{ is medium} \\
    0.95 & \text{if the value of slot } i \text{ is high} 
\end{cases}$$

We then assumed a transition model for the 3-slot dialogue MDP, solved it, and considered the optimal policy as the expert policy.

Finally, we varied the feature weights $w_1$ and $w_2$ from -50 to +50, learned various reward models for the expert, and found the optimal policy of each reward model, called the learned policy. For each state, we compared the learned action to the expert action, and counted the number of mis-matched actions.

Figure A.1 plots the number of the mis-matched actions. The part shown by the red arrow shows the space in which the reward models have an optimal policy that completely match to the expert policy. Therefore, the figure shows that there is a wide space with infinitive number of reward models whose policies completely matched with the expert policy. That is, IRL is an ill-posed problem.
A.2 LSPI-IRL

In this section, we present a variation of MDP-IRL algorithm, called LSPI-IRL, which is a model-free trajectory-based MDP-IRL algorithm. In LSPI-IRL, the candidate policies are estimated using the LSPI (least square policy iteration) algorithm [Lagoudakis and Parr, 2003]. In the model-free MDP problems, there is not a defined/learned transition model and the states are usually presented using features. Thus, model-free MDP algorithms are used for estimating the optimal policy of such MDPs. In this context, LSPI [Lagoudakis and Parr, 2003] is a common algorithm for estimating the optimal policy of such MDPs. We used LSPI in MDP-IRL described in Algorithm 7 to find the policy of each candidate reward model. As such, we have a variation of MDP-IRL algorithm called LSPI-IRL, described in Algorithm 10.

As stated earlier, in LSPI-IRL there is no access to a transition function but only the expert trajectories $D = (s_0, \pi_E(s_0), \ldots, s_{B-1}, \pi_E(s_{B-1}))$, where $B$ is the number of expert trajectories. In LSPI-IRL, we use LSTDQ (least-squares temporal-difference learning for the state-action value function), introduced in Lagoudakis and Parr [2003], to estimate candidate policy values $v^\pi$ and expert policy values $v^{\pi_E}$, shown in Equation (5.5) and in Equation (5.7), respectively. In LSPI-IRL, these estimated values are denoted by $\hat{v}^\pi$ and $\hat{v}^{\pi_E}$, respectively. Therefore, in IRL for POMDPs we maximize the margin:

$$d' = (\hat{v}_{s}^{\pi_E} - \hat{v}_{s}^{\pi_1}) + \ldots + (\hat{v}_{s}^{\pi_E} - \hat{v}_{s}^{\pi_t})$$

**Input:** Expert trajectories in the form of \( D = \{ (s_n, \pi_E(s_n), s'_n) \} \), a vector of features \( \phi = (\phi_1, \ldots, \phi_K) \), convergence rate \( \epsilon \), and maximum Iteration \( \text{maxT} \)

**Output:** Finds reward model \( R \) where \( R = \sum_i \alpha_i \phi_i(s, a) \), by approximating \( \alpha = (\alpha_1, \ldots, \alpha_K) \)

1. Choose the initial reward \( R^1 \) by randomly initializing feature weights \( \alpha \);
2. Construct \( D' \) by inserting \( R^1 \) in \( D = \{ (s_n, \pi_E(s_n), r^1_n, s'_n) \} \);
3. Set \( \hat{\pi}_1 = \{ \hat{\pi}_1 \} \) by finding \( \hat{\pi}_1 \) using LSPI and \( D' \);
4. Set \( \hat{X} = \{ \hat{x}^\pi_1 \} \) by finding \( \hat{x}^\pi_1 \) from Equation (A.9);
5. for \( t \leftarrow 1 \) to \( \text{maxT} \) do
6. Find values for \( \alpha \) by solving the linear program:
   \[
   \text{maximize } d^t = \left[ (\hat{x}^E - \hat{x}^\pi_1) + \ldots + (\hat{x}^E - \hat{x}^\pi_t) \right] \alpha;
   \]
   subject to \( 0 \leq |\alpha_i| \leq 1 \);
   and \( \hat{x}^E \alpha - \hat{x}^\pi_1 \alpha > 0 \ \forall \pi_l \ \text{1 \leq l \leq t} \);
7. Update \( D' \) to \( D' = \{ (s_n, \pi_E(s_n), r^t_n + 1, s'_n) \} \) using \( R^{t+1} = \phi \alpha \);
8. if \( \max_i |\alpha_i^t - \alpha_i^{t-1}| \leq \epsilon \) then
9.   return \( R^{t+1} \);
10. else
11.   Find \( \hat{\pi}_{t+1} \) using LSPI and the updated trajectories \( D' \)
12.   \( \hat{\Pi} = \hat{\Pi} \cup \{ \hat{\pi}_{t+1} \} \);
13.   Set \( \hat{X} = \hat{X} \cup \{ \hat{x}^\pi_{t+1} \} \) by calculating \( \hat{x}^\pi_{t+1} \) from Equation (A.9);
14. end
15. end

Lagoudakis and Parr [2003] showed that the estimate of state action values \( \hat{Q}^\pi(s, a) \), can be calculated as: \( \hat{Q}^\pi(s, a) = \phi(s, a)^T \omega^\pi \). Therefore, we have:

\[
\hat{V}^\pi(s) = \phi(s, \pi(s))^T \omega^\pi
\]

Using the vector representation, we have:

\[
\hat{v}^\pi = \Phi^\pi \omega^\pi
\]

where

\[
\Phi^\pi = \begin{pmatrix}
\phi(s_0, \pi(s_0))^T \\
\ldots \\
\phi(s_{B-1}, \pi(s_{B-1}))^T
\end{pmatrix}
\]
and $\omega^\pi$ is estimated by [Lagoudakis and Parr, 2003] as:

$$\omega^\pi = (B^\pi)^{-1} b$$  \hfill (A.2)

in which

$$B^\pi = \sum_{(s, \pi_E(s), s')} \phi(s, \pi_E(s)) (\phi(s, \pi_E(s)) - \gamma \phi(s', \pi(s')))^T$$

and

$$b = \sum_{(s, \pi_E(s))} \phi(s, \pi_E(s)) r(s, \pi_E(s))$$

Note that Lagoudakis and Parr [2003] used a slightly different notations than us. For the actions in data, they use $a_n$, however, we use $\pi_E(s_n)$, since we assume that the actions in data are the expert actions.

Using matrix representation for $B^\pi$ and the vector representation for $b$, we have:

$$B^\pi = \Phi^T (\Phi - \gamma \Phi'^\pi)$$  \hfill (A.3)

and

$$b = \Phi^T r$$  \hfill (A.4)

where $\Phi$ is a $B \times K$ matrix defined as:

$$\Phi = \begin{pmatrix} 
\phi(s_0, \pi_E(s_0)^T \\
\vdots \\
\phi(s_{B-1}, \pi_E(s_{B-1}))^T 
\end{pmatrix}$$

and $\Phi'^\pi$ is a $B \times K$ matrix defined as:

$$\Phi'^\pi = \begin{pmatrix} 
\phi(s'_0, \pi(s'_0))^T \\
\vdots \\
\phi(s'_{B-1}, \pi(s'_{B-1}))^T 
\end{pmatrix}$$

and $r$ is the vector of size $B$ of rewards:

$$r = \begin{pmatrix} 
r_0 \\
\vdots \\
r_{B-1} 
\end{pmatrix}$$

Moreover, $r$ can be represented using a linear combination of features:

$$r = \Phi \alpha$$  \hfill (A.5)
Having Equation (A.3), Equation (A.4), and Equation (A.5) in Equations (A.2), we can find the vector $\omega^\pi$, define as:

$$\omega^\pi = B^\pi b$$

$$= B^\pi \Phi^T r$$

$$= (\Phi^T (\Phi - \gamma \Phi^\pi))^{-1} \Phi^T \Phi \alpha$$  \hspace{1cm} (A.6)

Having Equation (A.6) in Equation (A.2), we have:

$$\hat{v}^\pi = \Phi^\pi \omega^\pi$$

$$= \Phi^\pi (\Phi^T (\Phi - \gamma \Phi^\pi))^{-1} \Phi^T \Phi \alpha$$  \hspace{1cm} (A.7)

Similar to Equation (5.5), $\hat{v}^\pi$ can be represented using feature weights $\alpha$ and an estimate for feature expectation, denoted by $\hat{x}^\pi$:

$$\hat{v}^\pi = \hat{x}^\pi \alpha$$  \hspace{1cm} (A.8)

Comparing Equation (A.8) to Equation (A.7), we have the estimate of $\hat{x}^\pi$:

$$\hat{x}^\pi = \Phi^\pi (\Phi^T (\Phi - \gamma \Phi^\pi))^{-1} \Phi^T \Phi$$  \hspace{1cm} (A.9)

Similarly, the expert policy $\hat{v}^\pi_E$ can be represented using feature weights $\alpha$ and an estimate for expert feature expectation, denoted by $\hat{x}^\pi_E$:

$$\hat{v}^\pi_E = \hat{x}^\pi_E \alpha$$  \hspace{1cm} (A.10)

And the estimate of feature expectation for expert policy, $x^E$, can be calculated as:

$$\hat{x}^\pi_E = \Phi^\pi_E (\Phi^T (\Phi - \gamma \Phi^\pi_E))^{-1} \Phi^T \Phi$$  \hspace{1cm} (A.11)

Algorithm 10, called LSPI-IRL, is similar to the MDP-IRL algorithm, described in Algorithm 7. LSPI-IRL starts by randomly initiating values for $\alpha$ to generate the initial rewards $R^1$. The algorithm then constructs trajectories $D'$ by inserting rewards $R^1$ inside the expert trajectories. In this way, the estimate of policy of $R^1$, denoted by $\hat{\pi}_1$, can be found using $D'$ in LSPI. Then, $\hat{\pi}_1$ is used in Equation (A.9) to construct $\hat{x}^\pi_1$.

In the first iteration of LSPI-IRL, using linear programming, it finds values for $\alpha$ that maximizes $\hat{x}^\pi_E \alpha - \hat{x}^\pi_1 \alpha$. The vector of learned values for $\alpha$ makes a candidate reward function $R^2$ which is used for updating trajectories $D'$ to be used in LSPI for learning the candidate policy $\pi_2$. The candidate policy $\pi_2$ in turn introduces a new feature expectation $\hat{x}^\pi_2$ using Equation (A.9). This process is repeated: in each iteration $t$, LSPI-IRL finds rewards by finding values for $\alpha$ which makes the approximate value for policy $\hat{\pi}_E$, denoted by $\hat{x}^\pi_E \alpha$ better than any other candidate policy. This is done by maximizing $d^t = \sum_{i=1}^t \hat{x}^\pi_E \alpha - \hat{x}^\pi_1 \alpha$ for all $t$ candidate policies learned so far up to iteration $t$. In this optimization, we also constrain the value of the expert’s policy to be greater than that of other policies in order to ensure that the expert’s policy is optimal, i.e., the constraint in Line 9 of the algorithm.
A.2.1 Choice of features

Similar to the experiments in Chapter 6, we need to define features for representing the reward model. In the LSPI-IRL algorithm, the features are also used in the LSPI algorithm, for estimating the policies. In this section, we introduce three kinds of features which are used in our experiments of the following section on the 3-slot problem. These features include:

1. *binary* features,
2. *2-flat* features,
3. *state-action-wise* features,

in which the expressive power increases from the binary features (least expressive) to state-action-wise features (most expressive).

The binary features use a binary representation for slots. In binary features four indexes are used to show value of one slot, in which empty (0), low(1), medium(2), high(3), are respectively represented as 0001, 0010, 0100, and 1000. For instance, in the 3-slot problem, for the state 3 1 2, i.e., the first slot has high(3), the second has low(1), and the third has medium(2) confidence score, the binary representation is as follows:

```
1000 0010 0010
```

Then, we use more expressive features. That is, we use *2-flat* features to show the interaction across slots. The 2-flat features are represented as follows. First, every possible 2 combination of slots are chosen and then for each combination the flat representation is used. In flat representation the index value is represented using the binary representation. For instance for the given example in the 3-slot problem, 3 1 2, the combination of size 2 of slots becomes: 31 32 12. Then, for the flat representation, we need to index each value and then show the index in binary representation. In total, there are 16 combinations of size 2: These include: 00, 01, ..., 31, 32, 33, which we index them from 1 to 16. Thus, the index for 31, 32, 12 respectively is 14, 15, 7. Finally, the binary representation of each index respectively is:

```
001000000000000 010000000000000 000000010000000
```

The most expressive features are the state-action-wise features, as defined in Chapter 6. In state-action-wise features there is an indicator function for each state-action pair.
A.2.2 Experiments

We applied LSPI-IRL for learning a reward model of the expert policy in which the expert policy is a MDP policy (cf. Section 5.1). More specifically, the expert policy is the optimal policy of the reward model shown in Equation (A.1) in which the feature weights are set to $w_1 = +30$ and $w_2 = -60$.

Table A.1 shows the LSPI-IRL performance for the 3-slot problem using 500 expert trajectories used for training and testing. The experiments have been performed using the three different features introduced in the previous section. The results of the table are based on criterion 1 introduced in Section 6.3.4. That is, the percentage of the learned actions that matches to the expert actions.

First, the table demonstrates that using state-action-features LSPI-IRL can learn a reward model that completely accounts for the expert policy. Then, it shows that as the expressive power of features decreases, the LSPI-IRL performance decreases. The values in the parenthesis shows the size of features. As expected, the state-action-wise features have the largest size and they show the best performance, in terms of match to the expert policy, while the binary features with the smallest size shows the least performance.

<table>
<thead>
<tr>
<th>features</th>
<th>percentage of matched actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>state-action-wise(1024)</td>
<td>100%</td>
</tr>
<tr>
<td>2-flat(384)</td>
<td>90%</td>
</tr>
<tr>
<td>binary(96)</td>
<td>85%</td>
</tr>
</tbody>
</table>

**Table A.1:** The LSPI-IRL performance using three different features.
Bibliography


Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue (SIGdial’08), Columbus, Ohio, USA.


