Statistical Dialogue Modelling

Milica Gašić

Department of Engineering
and
Trinity Hall

University of Cambridge

This dissertation is submitted for the degree of

Doctor of Philosophy

January 2011

Declaration

This dissertation is submitted for the degree of Doctor of Philosophy (PhD) at the Department of Engineering, University of Cambridge. I declare that it is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text. When reference is made to the works of others, the extent to which that work has been used is indicated and acknowledged in the text and bibliography. Some of the material included in this thesis has been previously presented at international conferences [1, 2], in a peer-reviewed journal [3] and a technical report [4].

The length of this dissertation, including appendices, bibliography, footnotes, tables and equation symbols is approximately 50,000 words. This dissertation contains 59 figures and tables.

To my parents

Acknowledgements

I would like to thank my supervisor, Prof. Steve Young, for his devoted supervision throughout my PhD. His guidance has been invaluable for my research. He has always found time to meet me and respond to my questions and for that I am very grateful to him. I am also grateful to Carl Rasmussen for his insights into Gaussian processes and his suggestions. I would also like to thank Yaakov Engel for his help with the GP-Sarsa algorithm.

The Dialogue Systems Group has provided a highly motivating and pleasant environment in which to carry out research. For that and all the support that they have given me, I must thank the current and the previous members of the group: Blaise Thomson, Jost Schatzmann, Simon Keizer, Kai Yu, François Mairesse, Tomoki Toda, Fabrice Lefèvre, Filip Jurčíček and Jorge Prombonas. I am grateful to Janet Milne and Rachel Fogg for their excellent administrative support.

I would also like to thank other members of the Machine Intelligence Group, and in particular, Rogier van Dalen, Matt Seigel, Matt Shannon, Jurgen van Gael, Shakir Mohamed and Marc Deisenroth for numerous discussions.

Thanks go to Rogier van Dalen, Blaise Thomson, Matt Seigel, Simon Keizer, Kai Yu and Dan Greenfield for proof-reading various parts of my draft.

Finally, I am grateful to the EPSRC for funding my research via grant agreement EP/F013930/1. I also appreciate the financial support I received from Trinity Hall and ISCA.

Abstract

The partially observable Markov decision process (POMDP) has been proposed as a model for dialogue which is able to provide increased robustness to errors in understanding of speech, automatically optimise dialogue management behaviour and be amenable to adaptation for different user types. The POMDP-based approach to dialogue management maintains a distribution over every possible dialogue state, the *belief state*. Based on that distribution the system chooses the action that gives the highest expected reward, where the reward provides a measure of how good the dialogue is. The primary challenge, however, with the POMDP-based approach is the intractability of both maintaining the belief state and of optimising action selection. The Hidden Information State framework is a practical framework for building dialogue managers based on the POMDP approach. It achieves tractability by grouping the possible user goals into equivalence classes which then ensures that the belief state can be maintained tractably. It optimises the dialogue policy in a much reduced belief state space, the *summary space*.

In this thesis, a more efficient state representation is presented which includes the representation of logical complements of concepts in the user request. On the one hand, the representation supports more complex dialogues that include logical expressions. On the other hand, it enables a pruning technique to be implemented which is able to place a bound on the space. Thus, no limit is required on the length of the dialogue or on the number of different hypotheses that are received from the speech understanding module. More importantly, this enables building real-world dialogue systems with large domains.

This thesis also examines the potential for improving the action selection. Firstly, the problem of optimising action selection in the summary space is examined. A method is then proposed that guarantees selection of optimal back-off actions in the case when the selected action cannot be mapped back to the original belief state space. Secondly, this thesis investigates the use of Gaussian processes to approximate the highest expected reward that can be obtained for every belief state and system action. Approximating the function with a Gaussian process provides a posterior distribution of the function values given the prior distribution and some observations. It is shown here that an adequate prior speeds up the optimisation of action selection. The posterior also provides an estimate of the uncertainty, which enables rapid adaptation to different user profiles.

Overall, the methods proposed in this thesis make steps towards more flexible real-world spoken dialogue systems.

Notation

General:	
$P(\cdot)$	probability
$P(\cdot\|\cdot)$	conditional probability
$E(\cdot)$	expectation
$\mathbb{R}$	the set of real numbers
$\mathbb{N}$	the set of natural numbers
Markov decision process:	
a_t	the action at time step t (random variable)
$\mathcal{A}$	a set of actions
a	an element of $\mathcal{A}$
s_t	the state at time step t (random variable)
$\mathcal{S}$	a set of states
s, s'	elements of $\mathcal{S}$
r_t	the immediate reward at time step t (random variable)
r	an element of $\mathbb{R}$
$\mathcal{P}^a_{ss'}$	the transition probability, $P(s_{t+1} = s' \mid s_t = s, a_t = a)$
$\mathcal{R}^a_s$	the expected immediate reward, $E(r_{t+1} \mid s_t = s, a_t = a)$
π	a policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$
γ	the geometric discount factor
R^π_t	the discounted return for policy π at time step t (random variable)
$V^\pi(s)$	the Value function for policy π, $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$
$V(s)$	the optimal Value function
$Q^\pi(s, a)$	the Q-function for policy π, $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
$Q(s, a)$	the optimal Q-function
$N(s, a)$	number of times action a is taken in state s, $N : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{N}$
ϵ	the exploration factor
λ	the step size parameter

Partially observable Markov decision process:

o_t	the observation at time step t (random variable)
$\mathcal{O}$	a set of observations
o, o'	elements of $\mathcal{O}$
$\mathcal{P}_{so}$	the observation probability, $P(o_t = o \vert s_t = s)$
$b(s_t)$	belief state at time step t
$b(s_t = s)$	probability of being in state $s \in \mathcal{S}$ at time step t
$\mathcal{B}$	the belief space, $[0,1]^{\vert\mathcal{S}\vert}$
$\mathbf{b}$	an element of $\mathcal{B}$
$V(\mathbf{b})$	the optimal Value function, $V : \mathcal{B} \to \mathbb{R}$
$Q(\mathbf{b}, a)$	the optimal Q-function, $Q : \mathcal{B} \times \mathcal{A} \to \mathbb{R}$
$V^{\pi}(\mathbf{b})$	the Value function for policy π
Π_t	the set of all policy trees at step t
$\mathcal{V}_t$	the set of the Value functions for policy trees Π_t
$\mathcal{V}_t^{+}$	a pruned set of the Value functions
ν	threshold

POMDP-based dialogue modelling:

a_t^m	the system's action at dialogue turn t (random variable)
$\mathcal{A}^m$	the set of all system's actions
a_m	an element of $\mathcal{A}^m$
a_t^u	the user action at dialogue turn t (random variable)
$\mathcal{A}^u$	the set of all user actions actions
a_u, a'_u	elements of $\mathcal{A}^u$
s_t^u	the user goal at dialogue turn t (random variable)
$\mathcal{S}^u$	the set of user goals
s_u, s'_u	elements of $\mathcal{S}^u$
s_t^d	the dialogue history at dialogue turn t (random variable)
$\mathcal{S}^d$	the set of all dialogue histories
s_d, s'_d	elements of $\mathcal{S}^d$

Hidden Information State dialogue modelling:

p_t	the partition at dialogue turn t (random variable)
$\mathcal{P}_t$	the set of partitions at turn t
p	an element of $\mathcal{P}_t$
p'	an element of $\mathcal{P}_{t+1}$
η	a node in partition
h_t	the hypothesis at dialogue turn t (random variable)
$\mathcal{H}_t$	the set of hypothesis at turn t
h	an element of $\mathcal{H}_t$
h'	an element of $\mathcal{H}_{t+1}$
$\tilde{a}_u^i$	ith element in the N-best list of the user input
c_i	a confidence score associated with ith element in the N-best list
$\mathcal{M}(a'_u, p', a_m)$	item matching function $\mathcal{M} : \mathcal{A}^u \times \mathcal{P}_{t+1} \times \mathcal{A}^m \to \mathbb{R}$
ρ	a grounding state
Γ	the set of grounding states
$\tau(\rho, a_m, a'_u)$	grounding state transition function, $\tau : \Gamma \times \mathcal{A}^m \times \mathcal{A}^u \to \Gamma$
$\hat{\mathbf{b}}$	a summary point
$\hat{a}_m$	a summary action

Extended HIS state representation:

$\alpha = \beta$	an attribute-value pair corresponding to nodes α and β
χ	the pruning threshold (the maximum number of partitions)

Gaussian process reinforcement learning:

$\mathbf{x}$	a multidimensional continuous data point
$\mathcal{N}(\cdot, \cdot)$	a Gaussian distribution
$\mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}))$	a Gaussian process with mean $m(\mathbf{x})$ and kernel function $k(\mathbf{x}, \mathbf{x})$
$k(\mathbf{x}, \mathbf{x}; \Theta)$	a kernel function with parameters Θ
$k_{\mathcal{X}}(\mathbf{x}, \mathbf{x})$	a kernel function where $\mathbf{x} \in \mathcal{X}$
$f(\mathbf{x})$	the latent function
$\overline{f}(\mathbf{x})$	the mean of latent function $f(\mathbf{x})$
$cov(\mathbf{x}, \mathbf{x})$	the covariance function
ξ	the Gaussian noise, $\xi \sim \mathcal{N}(0, \sigma^2)$
y	a noisy function observation
$\mathbf{X}_t$	a vector of data points (states) up-to time step t
$\mathbf{y}_t$	a vector of observations up-to time step t
$\mathbf{K}_t$	the Gram matrix for data points $\mathbf{X}_t$
$\mathbf{K}_t(\Theta)$	a Gram matrix where the kernel function is parametrised with Θ
$\mathbf{k}_t(\mathbf{x})$	a vector of the kernel function values between $\mathbf{x}$ and data points $\mathbf{X}_t$

$\phi_i(\cdot)$	a kernel feature function where $i \in \{1, 2, \dots\}$
$\boldsymbol{\phi}(\cdot)$	the kernel feature vector
$\mathbf{B}_t$	a vector of belief state-action pairs up-to time step t
$\mathbf{r}_t$	a vector of observed rewards in data points $\mathbf{B}_t$
$\mathbf{q}_t$	a vector of the Q-function values in data points $\mathbf{B}_t$
$\Delta Q^{\pi}(\mathbf{b}, a)$	residual of the Q-function, $\Delta Q^{\pi}(\mathbf{b}, a) \sim \mathcal{N}(0, \sigma^2)$
$\boldsymbol{\Delta}\mathbf{q}_t$	a vector of the residual values in data points $\mathbf{B}_t$
$(\tilde{\mathbf{b}}, \tilde{a})$	a representative belief state-action pair
$\mathcal{D}$	the set of representative points – the *dictionary*
$\mathbf{g}_t$	a vector of coefficients
$\delta_a(a')$	the Kronecker δ-function $\delta_a(a') = 1$ for $a' = a$, 0 otherwise
Adaptation:	
$\mathcal{M}$	a model

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The Oxford dictionary defines a dialogue as "a conversation between two or more people". According to the same dictionary, a computer is "an electronic device which is capable of receiving information (data) in a particular form" and able "to produce a result in the form of information or signals". But is dialogue really restricted to humans, as suggested by the Oxford dictionary, or can it instead be viewed as an exchange of information in a human-like manner, for example via speech? Will computers one day be able to take part in a dialogue? This thesis develops techniques for building such a computer, with the aim that one day humans will be able to speak to them as they do among themselves.

A system that can maintain an intelligent conversation with a human, via speech, and in real time, is called a spoken dialogue system. In this thesis, the scope of the conversation will always be limited, and is called the dialogue domain. A special emphasis is placed on real-world spoken dialogue systems, which are systems where the domain is large enough to enable conversation with some practical use. There are many challenges that arise when scaling these systems to large domains, and this thesis aims to address these.

Real-world spoken dialogue systems have innumerable benefits. Using speech a computer interface facilitates swift, human-like acquisition of information. Desktop computers have already been replaced by more compact, easily portable laptops, which are in turn rapidly being made obsolete by significantly smaller and increasingly powerful smart phones. The ability to acquire information almost instantaneously, at any time, anywhere, in increasingly simplified ways has become paramount. By replacing the need for typing, a spoken dialogue system makes it possible for a user to obtain information while using their hands to perform some other task, which in many cases will be the user's *primary task*. An example of using dialogue systems for *secondary tasks* is the development of in-car spoken dialogue systems [5]. This domain is particularly challenging since it involves dealing with varying levels of noise and the robustness of such systems is crucial. Another challenge is to alleviate the cognitive load on the user since the user's ability to perform the primary task cannot be compromised.

The ability to sustain an intelligent conversation with a machine is hugely beneficial to time-consuming business transactions and aspects of everyday life – particularly those that currently rely on a human to provide information or require filling in forms on the web. Spoken dialogue technology is not only applicable to business and everyday use in developed countries, but is even more relevant for small businesses and communities in the developing world. Recent research [6] has found that in many rural areas of developing countries, more people regularly use mobile phones than can read or write. Access to automated systems facilitates the provision of useful information such as weather and agriculture reports, and can even serve as a portal for advertising local businesses. The benefits of such technology are already visible [6, 7].

For some time there has been significant interest in developing an open domain dialogue system that is able to handle arbitrary conversations. Ideally, such a system would understand and respond in the same way as a human might do, but has encyclopaedic knowledge. However, building such a system has proved challenging. There only currently exist text-based dialogue systems, so-called *chat-bots*, that produce seemingly natural sentences but fail to sustain an intelligent conversation[1].

While an open-domain dialogue system would be ideal, the majority of applications in fact involve dialogue with a limited domain. Examples include telephone banking, flight-booking, trouble-shooting and information-

[1]In the 2008 Loebner Prize competition, only three out of twelve judges mistook the winning chat-bot for a human. This was the best performance since the start of the competition in 1991, `http://www.loebner.net/Prizef/loebner-prize.html`.

providing systems [8, 9, 10]. The usability and reliability of such systems is crucial. Such systems have to be robust and be able to deal with a large number of different users, while remaining relatively easy to develop and extend [11].

Although human-computer dialogue has indeed been a topic of research for some time, the poor performance of the underlying speech recognition systems has hampered progress in the area. While the performance of speech recognition systems has steadily improved over recent decades and the performance in noise has dramatically improved, it is nowhere near human speech recognition performance [12]. This poses a significant challenge to deploying real-world spoken dialogue systems [9]. Even if accurate input is assumed, it is not trivial to determine optimal system behaviour [13]. There is therefore a need for a spoken dialogue systems framework that is able gracefully to deal with the imperfections of the speech recogniser, while simultaneously being able to sustain natural conversation [11].

Despite being marked as one of the main challenges for artificial intelligence six decades ago [14], human-computer dialogue systems have only in the last decade been able to benefit from data-driven techniques and machine learning algorithms. This has proved to be promising for improving robustness and extendibility of spoken dialogue technology [15].

The remainder of this section provides an overview of a typical spoken dialogue system's structure. Emphasis is placed on modularity and statistical approaches to building such a system, focusing on the dialogue manager in particular. Section 1.2 describes the motivation for the work presented in this thesis, with the main contributions outlined in Section 1.3. Finally, the structure of the remainder of the thesis is given in Section 1.4.

1.1.1 Spoken dialogue system structure

A spoken dialogue system enables human-computer interaction where the primary medium of both input and output is speech. In this thesis, dialogue is represented a sequence of dialogue turns in which each turn consists of a system utterance followed by a user utterance[1]. A spoken dialogue system

[1]There are, however, more complex ways of representing dialogue. In [16] a multi-party spoken dialogue system is built that supports both visual and speech inputs. Additionally, research in [17] investigated dialogue in which the turn is not restricted to one system and one user utterance but there is more flexibility in communication.

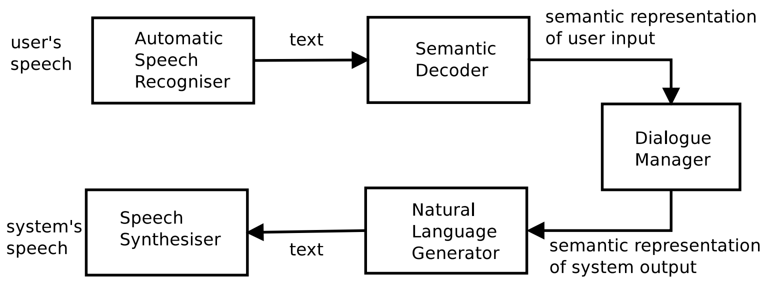

user's speech
Automatic Speech Recogniser
text
Semantic Decoder
semantic representation of user input
Dialogue Manager
semantic representation of system output
Natural Language Generator
text
Speech Synthesiser
system's speech

Figure 1.1: Spoken dialogue system structure.

normally consists of five components, as shown in Fig. 1.1. The first component is the speech recogniser, which transforms the user's speech into text. The text is then forwarded to the next component, the semantic decoder. The semantic decoder extracts a semantic representation of the user's intention, represented by semantic units called *concepts*. This is then forwarded to the dialogue management component, which selects the most appropriate system output, again in the form of semantic units. The system output is then passed to the next component – the natural language generator, which transforms the abstract semantic notation back into a text representation. The last component, the speech synthesiser, then converts this text into speech. See [18, 19] for a more detailed description.

In a statistical spoken dialogue system, the aim is to replace each of the afore-mentioned components with a statistical model that can be trained from data [20]. The overall goal is to build a data-driven dialogue system which improves over time and which users see as behaving in human-like way. That the components of such a system are based on probability distributions allows them to model the stochasticity that occurs in a dialogue. This approach allows the power of probability theory to be used in optimising the individual components of the system.

The system can also make use of probability theory to model the outputs of each component of the system as suggested in [21, 22, 23]. The outputs for each component are represented as a probability distribution over all the possible outputs that would have occurred in a non-statistical model. This has the potential to provide a more robust model of dialogue, since the information passed between components includes more information about the uncertainty in the interpretation of the user's input. Ideally, the automatic speech recognition (ASR) module produces a full probability distribution over every possible hypotheses of what the user said. The probability distribution is then forwarded to the semantic decoder, which produces a probability distribution over all possible semantic concepts. This is finally propagated to the dialogue manager. In this way, the dialogue manager is able to make use of the alternative options in the input to formulate an answer for the user depending on the uncertainty conveyed in the probability distribution. This approach is particularly useful in noisy environments, when the most-likely input might not be correct. In practice, the probability distributions are too complex to be represented in full and so an approximation is used instead. The probability distributions are normally approximated by a lattice [24] or an N-best list [25] of hypothesised outputs with associated confidence scores [26]. A confidence score gives an estimate of how likely a hypothesised output is, given the input and it can be provided at different levels of granularity. In terms of the ASR and semantic decoding, utterance-level and concept-level confidence scores will be assumed respectively.

1.1.2 Dialogue management

The central component of a spoken dialogue system is the dialogue manager. It is responsible for selecting the system responses – *actions* – and governs the flow of the dialogue. This thesis will assume that the dialogue manager can be decoupled into two components – that which keeps track of the dialogue state, called the *dialogue model*, and that responsible for selecting the system's action based on the dialogue state, called the *action selection* component. A mapping between dialogue states and system actions is called a dialogue *policy*. As was already outlined in the previous section, learning from data is capable of making dialogue systems more human-like and also reduces the development cost of hand-crafting every possible dialogue scenario.

All components of a spoken dialogue system can in principle be estimated using supervised learning techniques.[1] Speech recognisers normally use a type of statistical model called a hidden Markov model (HMM), which is trained on annotated speech data [28]. In a similar way, the speech syn-

[1]Refer to [27] for an overview of statistical models.

thesiser can also be based on HMMs [29]. Semantic decoding is often hand-crafted, but it can also be seen as a classification task that can be implemented with, for example, support vector machines [30]. Similarly, natural language generation is normally hand-crafted, but can be modelled using Bayesian networks trained from data [31]. Dialogue management could be seen as a classification task in which an appropriate dialogue action is taken for each dialogue state. Such a classifier can easily be trained from a corpus of annotated data. In such an approach, the action that is taken by the classifier in each dialogue state resembles the corresponding action seen in the corpus for that particular state. However, the actions that are taken should not merely imitate the actions seen in a given corpus, but should ideally lead to a successful dialogue. For this reason, it is more appropriate to view dialogue as a long term decision process, and consequently optimise the actions it takes according to the overall success of the dialogue. This is therefore a planning problem and one way of obtaining optimal solutions automatically is via reinforcement learning [32].

In the reinforcement learning approach, the dialogue is modelled in terms of the dialogue state, the system's action and the reward, the measure of dialogue success [33]. If the current dialogue state depends only on the previous dialogue state, the dialogue can be modelled as a Markov decision process (MDP) [32]. This makes it possible to determine the optimal policy – that which yields the highest reward, using MDP learning algorithms. When using a probability distribution over possible utterances, the equivalent model is the partially observable Markov decision process (POMDP) [34]. Then, instead of maintaining the dialogue state in every dialogue turn, a distribution of possible dialogue states, the *belief state*, is maintained throughout the dialogue. This makes it possible for the optimal actions that the system takes to be based not only on the overall dialogue success, but also on the differing levels of uncertainty in the dialogue. Formal definitions of Markov decision processes and partially observable Markov decision processes, and an overview of their use in statistical dialogue modelling will be given in Section 2.2.

1.2 Motivation

Modelling dialogue as a partially observable Markov decision process (POMDP) can achieve robustness in the face of understanding errors as it models uncertainty in the dialogue state via maintaining a belief state. However, a significant challenge is the tractability of both the dialogue modelling component and the action-selection component. The development of an approximation framework that can retain the benefits of this model for a real-world dialogue task, while at the same time remaining tractable both in the belief state estimation and action-selection, is the core topic of this thesis.

Using statistical approaches to dialogue management provides a framework where other important aspects of dialogue modelling can be explored. This includes the topic of adaptation. More specifically, different users perceive and value dialogue differently. It is therefore important to ensure that a dialogue system can not only learn from data, but also be able to adapt quickly to different user types.

1.3 Contribution

The work presented here has been carried out on a special form of statistical dialogue system, called the Hidden Information State (HIS) dialogue system [35], and was motivated by the limitations of the original design which were identified in a user study. The contribution of this thesis may be conveniently divided into two parts – the contribution to state representation in a POMDP dialogue manager, and the contribution to policy optimisation.

The first contribution concerns the representation of the dialogue state in a way that supports complex dialogues. Although this work extends previous work on the HIS framework, the techniques used are applicable to any POMDP-based system which has a similar joint representation of concepts in the dialogue state or in a part of the dialogue state.

The second contribution may further be subdivided into two parts. The first concerns an investigation of the issues that arise in policy optimisation in a reduced dialogue space, the *summary space* [36]. This is relevant because POMDP dialogue managers typically reduce the dialogue state space to a smaller scale space in order to perform the policy optimisations tractably.

The second part investigates policy optimisation and uses a Bayesian approach to speed up the learning process and enable adaptation to different user profiles. This part of the contribution is independent of the adopted HIS framework.

More details on the primary findings of this thesis are as follows:

1.3.1 Effective handling of the dialogue state

The information that is included in the dialogue state is crucial for successful dialogue system operation. As will be explained in detail in Section 2.2.5, intractability is the main obstacle to a wider application of the POMDP-based approach. In fact, the difficulty in updating the distribution over all possible dialogue states normally places a constraint on the complexity of the supported dialogue domain, the length of the N-best list of user inputs as well as the length of the dialogue itself. In this thesis a more flexible representation of dialogue state is presented which supports more complex dialogues, while at the same time allowing a pruning mechanism to be implemented such that neither the length of the N-best user input list nor the length of the dialogue is constrained.

1.3.2 Policy optimisation in reduced dialogue state space

State of the art methods for policy optimisation in POMDP-based dialogue management rely on reducing the dialogue state space and performing policy optimisation in the resulting reduced space, called the *summary space*. While such methods enable tractability, various heuristics are required in order to map a summary action into an appropriate dialogue action in the original space. Not every summary action is applicable to every possible belief state so back-off action-selection strategies are required. The effects of back-off selection are investigated here and a strategy that is guaranteed to perform optimally is presented. An important observation that is made here is the reduced ability of standard non-parametric algorithms to deal with the expansion of the summary space.

1.3.3 Gaussian processes in POMDP-based dialogue management policy optimisation

Non-parametric policy optimisation solutions require many training iterations to reach convergence even in a reduced dialogue state space. Parametric methods can speed up the learning process but necessitate a manual choice of basis functions and provide the optimal solution only within the given basis. This thesis investigates the use of Gaussian processes as a non-parametric Bayesian approach for function approximation [37]. This approach incorporates a prior probability which speeds up the policy optimisation process. The approach also provides a measure of the uncertainty during policy optimisation, which is shown to be particularly useful for dialogue management, since it enables further acceleration of the learning process in an active learning setting. In addition, the measure of uncertainty can be used as a policy selection criterion for rapid adaptation to different user profiles.

1.4 Thesis structure

The next chapter offers an introduction to spoken dialogue systems research, with particular attention paid to statistical approaches. Chapter 3 provides an overview of the Hidden Information State dialogue system, which is the statistical dialogue system framework studied in this thesis. In Chapter 4, an extension to the dialogue state representation is described, which enables tractability in more complex dialogue domains. Chapter 5 discusses policy learning and its challenges. In Chapter 6, Gaussian processes for dialogue policy optimisation are introduced with particular emphasis on their ability to represent uncertainty in policy learning, as well as their ability to accelerate the learning process. Chapter 7 describes how Gaussian processes in dialogue management can be used to support adaptability to different user profiles. Finally, Chapter 8 presents conclusions and discusses possible avenues of fruitful future research.

Chapter 2

Background

2.1 Introduction

The theory of spoken dialogue systems covers a large research area with many distinct topics [18]. In order to simplify the exposition, this chapter will focus on providing an introduction to the main topics used in this thesis. The chapter will thus focus on task-oriented, limited domain, statistical dialogue systems that can be used for real-world applications.

The first section of the chapter discusses the complexity of the domain required for a real-world spoken dialogue system. This is followed by a discussion of various rule-based methods used for building these systems in Section 2.1.2. Some deficiencies of these methods will be highlighted and provide a motive for the statistical approaches to dialogue management presented later. Section 2.2 gives a detailed description of how reinforcement learning is used for dialogue management, both when dialogue is modelled as a Markov decision process and a partially observable Markov decision process. This description is accompanied by a review of spoken dialogue systems that are based on these models. When using these models, it is useful to have a simulator of the way users behave in a dialogue, called a *user simulator*. Section 2.3 explains the importance of user simulation and describes the basic theory of this topic. Section 2.4 concludes the chapter with a discussion on evaluation methods.

2.1.1 Domain size of a real-world dialogue manager

Before an overview of statistical approaches for dialogue management is given, it is important to discuss what is meant here by a real-world task-oriented dialogue system. As noted in the introduction, the key feature of a real-world spoken dialogue system is that it should be useful in a practical situation, for example in replacing a human operator in a call centre. The system can be restricted to a specific task, for example providing information about a restaurant [38], booking a flight [8], or giving instructions to a user on how to fix a modem [10]. Such dialogue systems typically have access to a database which contains all the information about the particular domain [18]. The size of the domain can thus be defined in terms of the size of the underlying database, which in turn can be described in terms of the number of entities, attributes and values. A real-world system should be able to handle at least 100 entities, around 10 attributes, and hundreds of values. For the purposes of this thesis, any system which can do this will considered to be a real-world system, although there are clearly many tasks which require the system to scale further.

Good examples of real-world dialogue managers are the systems built for the Spoken Dialogue Challenge [39]. The task for these systems was to provide bus information in Pittsburgh – users should be able to call the service and ask which buses go from one stop to another and at what times [9]. The database consists of more than $300,000$ bus stops, along with the bus routes and arrival times for each stop. In the Spoken Dialogue Challenge several systems were built for this task with the aim of evaluating them in interaction with large number of real users. More information about the systems and the outcomes of the challenge can be found in [40].

2.1.2 Rule-based approaches to dialogue management

As outlined in Section 1.1.2, the role of the dialogue manager is to control the flow of the dialogue. A simple way to achieve this is to define a set of rules that the system follows during the course of the dialogue. In such a framework, the dialogue manager typically tries to maintain control of the dialogue by asking the user questions which the user then answers. Such an approach is called a *system-directed* dialogue [18]. The dialogue model in such systems is usually a finite state automaton (FSA) where each node of the FSA represents a dialogue state with the associated system questions and the arcs denote possible user answers [41]. Current commercially deployed dialogue managers are based on a similar structure – *call-flow* [42, 43]– which makes use of high-level specification languages such as VoiceXML [44, 45] to implement the functional specification and the detailed design of the interaction. Researchers have noted several limitations to this approach [11]. Perhaps, the most important is that the system design depends heavily on the domain. As a result, it is difficult to extend the system – every extension to the database requires arcs and nodes to be added to the call-flow manually. It also clearly has no capacity automatically to improve the dialogue manager's behaviour. Finally, these systems are sensitive to speech-understanding errors and require hand-crafted error handling schemes to deal with these errors, typically in the form of a dedicated error-handler.

An alternative approach to rule-based system design is known as *frame-filling* or *form-filling* [46]. A distinctive feature of this approach is that it decouples the rules dealing with user input from those controlling the dialogue flow. The user goal is described in terms of *slots*, which represent a concept that a user can talk about, and *fillers*, which represent a piece of information that can be obtained from the user input for that particular concept. Over the dialogue, a collection of slots, called a *form* or *frame*, will be filled with the values provided by the user. Note that if these slots and fillers correspond to the attributes and values of the underlying database, the complexity of these systems can be expressed in terms of the number of slots and the number of fillers they support.

The rule-based element of this approach arises when defining the dialogue flow. It is represented as an FSA, where the transitions govern which actions to take given how populated the current form is. This approach allows the user to specify their request and not necessarily directly follow the system's questions. This is example of an approach that supports *user initiative* during the dialogue [18]. The form-filling approach was further extended in the *agenda-based* dialogue management framework to support more complex dialogues. In the agenda-based approach, the dialogue consists of sub-dialogues and each sub-dialogue has an agenda that directs its flow [47]. While this approach is more flexible, it does not support automatic optimisation of the dialogue manager and does not replace the need for a separate error handler that deals with speech understanding errors.

The role of an error handler is to detect speech-understanding errors and to provide a strategy to deal with them [48]. One way of detecting errors is to check whether the inputs are reasonable given the state of the system and system action. However, this approach can easily fail because the system state is built assuming the past inputs were correct. One simple way to deal with this is to have a very conservative dialogue manager behaviour which confirms almost every user input, thus preventing an error from occurring. Such policies are typically very unnatural and annoying to the user [11].

Another way of detecting errors is to use the confidence measure assigned to each input by the speech-understanding unit. If the confidence is above a certain threshold, the input is accepted, and otherwise it is removed. This does provide a more refined correction strategy but its effectiveness depends vitally on the confidence measure being reliable, which is often not the case [43]. Even when it is reliable, this approach will fail when the same mis-recognitions reoccur in a dialogue [49]. Repeatedly rejecting the input leads to an infinite loop, since the system keeps asking the same question and the user gives the same answer.

2.1.3 Statistical approaches to dialogue management

When building a commercially deployed system one usually starts with a functional specification of the desired dialogue management behaviour and builds the system to match it. Once the basic behaviour is implemented, the system is tested with users and then manually adjusted to improve user satisfaction. The process is repeated iteratively until a suitable level of user satisfaction is reached [50].

In principle, supervised learning techniques offer a way of learning the optimal dialogue management behaviour directly from data when given a large enough dialogue corpus. This allows the dialogue manager to be portable across different domains and to be easily extendible. However, the dynamic nature of dialogue results in large difficulties when applying data-driven techniques. Dialogue domains are usually at least exponential in the number of distinct instances they can generate. Even a very large dialogue corpus would represent only a tiny fraction of the total set of plausible dialogues. As a result, supervised learning faces severe sparsity issues and a significant amount of abstraction is needed to bound the space of behaviour that can be learnt. Even if the system's behaviour can be learnt in a supervised fashion, it would be restricted to imitating form of particular behaviour at a particular turn. There is no guarantee that such behaviour would lead to a successful dialogue [51].

An alternative is to use reinforcement learning, where the dialogue is modelled as a sequential decision process and the dialogue management behaviour is optimised with respect to an objective measure of dialogue performance [33]. In contrast to the supervised learning approach, where the dialogue manager's behaviour is only confined to one that occurs in the corpus, a dialogue manager using reinforcement learning can explore all possible behaviour. It is therefore able to choose a strategy which optimises the overall performance as defined by the objective measure. The next section provides an introduction to reinforcement learning theory.

2.2 Reinforcement learning in dialogue management

Reinforcement learning is a machine learning method whereby the agent (the machine) learns from interaction with the environment [32]. The machine has a perception of the environment, which is represented by a *state*. It interacts with the environment by taking *actions* and receiving *rewards*. The aim is to take actions that lead to the highest expected long-term reward. In principle, the state the machine is in depends on all previously visited states, as well as the actions it has taken. However, optimising an action based on all previously visited states and actions is intractable. Accordingly, the state is assumed to be satisfy the *Markov property* – it depends only on the previous state.

The remainder of this section explains the Markov decision processes and the way in which it can serve as a model for dialogue. It then introduces an extension called the partially observable Markov decision process and discusses its use in dialogue modelling.

2.2.1 Markov decision process

A Markov decision process (MDP) starts with the assumption of a set of states that a machine can be in at every particular time step, $\mathcal{S}$, and a set of actions, $\mathcal{A}$, that it can take in these states. At each time step t

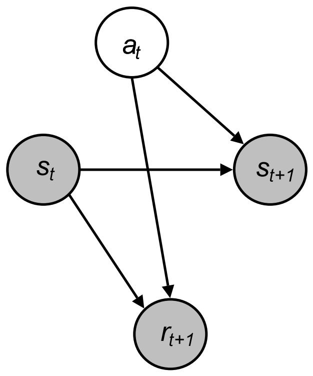


Figure 2.1: Markov decision process.

the machine is in a state s_t and takes an action a_t. It will then make a transition to the next state s_{t+1} and receive a reward r_{t+1}. The transition probability will depend only on the previous state and the action taken, $\mathcal{P}^a_{ss'} = P(s_{t+1} = s'|s_t = s, a_t = a)$ and the system therefore satisfies the Markov property. Note that the transition probabilities allow for either deterministic or stochastic transitions. In a similar way, the reward that is obtained is functionally dependent only on the state and the action taken $r_{t+1} = r(s_t = s, a_t = a)$ and takes values $r \in \mathbb{R}$. As an alternative one can also consider the reward to be a stochastic process, in which case the expected value of the immediate reward in time step $t+1$ is $\mathcal{R}^a_s = E(r_{t+1}|s_t = s, a_t = a)$. One way of visualising the Markov decision process is to represent it as a dynamic Bayesian network where states and rewards are fully observable. An example of part of the MDP network is given in Fig. 2.1, where shaded nodes represent observable variables.

In an MDP model, decisions on which action to take are determined by a *policy*, π, which is a mapping from the states of the system to the possible actions, $\pi : \mathcal{S} \rightarrow \mathcal{A}$. The *discounted return*, R^π_t, for a particular policy π is the sum of the discounted rewards that the policy π achieves from time t:

$$R^\pi_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}, \tag{2.1}$$

where, for each i, action $a_{t+i} = \pi(s_{t+i})$ is taken, r_{t+i+1} is the obtained reward and γ is a geometric discount factor $\gamma \in [0, 1]$. If the state transitions are random and the immediate reward is a random process, then the discounted return is also a random process.

The *Value function* $V^{\pi}(s)$ for each state s is defined as the expected discounted return in state s when a system follows policy π, $V^{\pi} : \mathcal{S} \rightarrow \mathbb{R}$:

$$V^{\pi}(s) = E_{\pi}\left(R_t^{\pi}|s_t = s\right) = E_{\pi}\left(\sum_{i=0}^{\infty} \gamma^i r_{t+i+1}|s_t = s\right), \tag{2.2}$$

where expectation E_{π} is calculated over all possible state sequences that can be generated with policy π.

In a similar way, the *Q-function* $Q^{\pi}(s, a)$, for each state s and action a, is defined as the expected discounted return that is obtained when action a is taken in state s and the policy π is followed from then on, $Q^{\pi} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$:

$$Q^{\pi}(s, a) = E_{\pi}\left(R_t^{\pi}|s_t = s, a_t = a\right) = E_{\pi}\left(\sum_{i=0}^{\infty} \gamma^i r_{t+i+1}|s_t = s, a_t = a\right), \tag{2.3}$$

where again expectation E_{π} is calculated over all possible state sequences that can be generated with policy π.

A discounting factor of less than 1 is normally used to favour policies that generate high rewards sooner rather than later in the decision process. A discounting factor equal to 1 can only be used in tasks that are certain to finish in a finite number of steps, called *episodic tasks*. Dialogue management is one example of an episodic task.

From Eq. 2.2 and 2.3 the following relation holds between the Value function $V^{\pi}(s)$ and the Q-function $Q^{\pi}(s, a)$ for the same policy π:

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s)). \tag{2.4}$$

The aim of reinforcement learning is to obtain the optimal policy, *i.e.,* the policy that maximises the Value function. Assuming a finite state space $\mathcal{S}$, the exact solution to the optimal Value function[1] is given by the Bellman optimality equation [52]:

$$V(s) = \max_a \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \cdot (\mathcal{R}_s^a + \gamma V(s')). \tag{2.5}$$

[1]Strictly, the optimal Value function should be denoted as $V^{\pi^*}(s)$, where π^* is the optimal policy. In order to keep the notation simple, the policy marker is dropped. Note, additionally, that for one optimal Value function there exist many optimal policies and vice versa [32], which further justifies the adopted notation.

In a similar way, the optimal Q-function is expressed by

$$Q(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \cdot (\mathcal{R}^a_s + \gamma \max_{a'} Q(s',a')). \tag{2.6}$$

The optimal Q-function $Q(s,a)$ and the optimal Value function $V(s)$ are related by

$$V(s) = \max_a Q(s,a). \tag{2.7}$$

The optimal policy can be derived from the optimal Value function

$$\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_s + \gamma V(s')], \tag{2.8}$$

or from the optimal Q-function

$$\pi(s) = \arg\max_a Q(s,a). \tag{2.9}$$

In order to solve Eq. 2.5 directly, several assumptions are needed. Firstly, as mentioned before, the Markov property has to be satisfied. It can be very difficult to build a state that on the one hand satisfies the Markov property and on the other hand is simple enough allow for learning to be performed. Secondly, the dynamics of the environment, namely the transition probabilities and the reward, need to be known. Finally, sufficient computational resources are required to calculate the solution. Although the dialogue state can be approximated, the reward can be heuristically defined and the dynamics can be hand-crafted or estimated from data. Obtaining the solution directly from the Bellman equation is normally intractable for large state spaces [32].

2.2.2 MDP policy optimisation

Solutions to the problem of MDP policy optimisation are often separated into model-based and model-free methods. Model-based methods assume that the dynamics of the environment are known, *i.e.,* the transition probabilities and the reward function, so that the policy can be optimised *off-line* – without direct interaction with the environment. Dynamic programming [52] is an example of a model-based method. Model-free methods learn through direct interaction *on-line* in a (possibly simulated) environment without assumptions about the underlying model of the environment. Monte

Carlo methods and temporal-difference learning are examples of model-free methods [32]. Model-free approaches often use ϵ-greedy learning, which involves *exploration* – taking a random action to explore the state space with probability ϵ, and *exploitation* – taking the action according to the best current policy with probability $1 - \epsilon$.

Algorithms that solve MDPs can also be divided into *on-policy* and *off-policy* methods. On-policy methods use the current best estimate of the optimal policy for exploitation while re-estimating it at the same time. *Off-policy* methods, on the other hand, estimate the optimal policy while exploiting another policy which may be suboptimal. The main advantage of on-policy methods is that they accumulate larger reward during the training and are faster in convergence to the optimal policy and thus more useful in direct on-line learning. Off-policy methods are typically used when there is a corpus of the machine-environment interaction available, generated with a suboptimal policy. Off-policy can then use the corpus to learn the optimal policy. However, it requires that every state action pair appears in the corpus, otherwise further approximations are needed [32].

Some of the most common algorithms used for MDP policy optimisation will now be presented. The exposition is based on the description given in [32], where the interested reader can also find a discussion on the convergence properties of these algorithms.

The *value iteration* algorithm is a commonly used dynamic programming algorithm for obtaining the optimal Value function (see Algorithm 1). Before the algorithm is explained, it is important to discuss the concepts of *stationary* and *non-stationary* policies.

The Value function as defined in Eq. 2.2 gives the same value for a particular state irrespective of the time step when the machine is in that state. A Value function that has such a property is called *stationary.* Its respective policy is also stationary – it does not depend on time, but for a particular state always proposes the same action.

A Value function that gives different values for the same state, depending on the time step the machine is in, is called a *non-stationary* Value function. A policy associated with such a Value function is a non-stationary policy – it gives different actions for the same state depending on the time step. If the MDP model (the transition probabilities and the expected reward) does not change with time, it can be shown that there is always an optimal policy which is stationary [32].

The value iteration algorithm re-estimates the optimal Value function at every time step. During the process of optimal Value function estimation, the optimal Value function in one state at one time step differs from the estimate at another time step for the same state. Thus, during the value iteration process, the Value function that is being estimated is non-stationary and it converges to a stationary Value function with enough iterations. The value iteration algorithm makes use of the recursive property of the Bellman equation (Eq. 2.5) to relate optimal Value function estimates between between two consequent time steps. If 1-step denotes the one time step to the final state, then the Value function in 1-step for some policy π in state s is simply the reward associated with action a when it is taken in state s, where action a is determined by policy π, $a = \pi(s)$. In general, the Value function of policy π in state s in t-step is the reward obtained by taking action a, where $a = \pi(s)$, and the discounted Value function in $(t-1)$-step of policy π in the next state. Because of this recursive relationship it is possible to initialise the Value function arbitrarily for 1-step and iteratively to update and maximise the Value function for each subsequent step until the difference between the t-step and the $(t-1)$-step Value functions falls below a given threshold.

Algorithm 1 Value iteration

```
1: for all $s \in \mathcal{S}$ do
2:     $V(s) \leftarrow$ arbitrary
3: end for
4: repeat
5:     for all $s \in \mathcal{S}$ do
6:         $v \leftarrow V(s)$
7:         $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_s + \gamma V(s')]$
8:         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
9:     end for
10: until $\Delta < \theta$
11: for all $s \in \mathcal{S}$ do
12:     $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_s + \gamma V(s')]$
13: end for
```

The Monte Carlo control algorithm (Algorithm 2) is a Monte Carlo method commonly used in episodic tasks. In these tasks one defines a start state and a collection of terminal states. An episode is then a sequence of states from the initial to a terminal state, generated in interaction with the environment. The Monte Carlo control algorithm starts by taking actions according to an arbitrary policy and generating episodes. For each episode, the sequence of states visited and actions taken is recorded. At the end of the dialogue, the discounted return[1] R is computed for each state and action pair and used to update the corresponding $Q(s,a)$ value. The updated value should reflect the average discounted return over all dialogues. The algorithm also maintains a count of the total number of times, $N(s,a)$, that each action pair has been visited. The policy is updated after each episode to select the actions that have accumulated the highest reward for each state. During an episode, however, the policy stays constant. This algorithm is particularly useful for policy optimisation of dialogue managers since a dialogue can be regarded as an episode generated in interaction with a user as long as the user or an independent evaluator gives a reward at the end of the dialogue.

Algorithm 2 Monte Carlo control algorithm

```
1:  for all $s \in \mathcal{S}$,$a \in \mathcal{A}$ do
2:    $Q(s,a) \leftarrow$ arbitrary
3:    $N(s,a) \leftarrow$ arbitrary
4:    $\pi(s) \leftarrow$ arbitrary
5:  end for
6:  repeat
7:    Generate an episode using exploitation of policy $\pi$ and exploration
      with probability $\epsilon$ i.e. $\epsilon$-greedily and record $(s,a)$ that appeared
8:    for all $(s,a)$ pairs appearing in the episode do
9:      $R \leftarrow$ discounted return following the first occurrence of $(s,a)$
10:     $Q(s,a) \leftarrow \frac{Q(s,a)*N(s,a)+R}{N(s,a)+1}$
11:     $N(s,a) \leftarrow N(s,a)+1$
12:   end for
13:   for all $s$ in the episode do
14:     $\pi(s) = \arg\max_a Q(s,a)$
15:   end for
16: until convergence
```

Temporal-difference (TD) learning represents another class of MDP policy optimisation algorithms. TD learning combines the ideas of Monte Carlo and dynamic programming methods. As with Monte Carlo, learning is per-

[1]Since this is an episodic task the return does not need to be discounted.

formed from experience at the level of an episode, but the Value function or the Q-value estimates are updated at each step, upon a difference in the obtained and expected rewards – the *temporal difference*. This entails a recursive relation similar to that used in dynamic programming algorithms. Examples of TD algorithms given here are Q-learning and Sarsa.

The Q-learning algorithm is an example of an off-policy TD learning (see Algorithm 3). The update is the temporal difference expressed as the difference between the discounted maximal future prediction and the current Q-value estimate deducted for the observed reward: $\gamma \max_{a'} Q(s', a') - (Q(s,a) - r)$ (see line 7 in Algorithm 3). The step size parameter λ, $0 \leq \lambda \leq 1$ controls how much the current temporal difference element contributes to the Q-value estimation. Q-learning is an off-policy method since one policy is followed while another is being estimated. This is due to the maximisation in the temporal difference.

Algorithm 3 Q-learning

```
1: Initialise $Q(s,a)$ arbitrary for every $(s,a)$ pair
2: for all episode do
3:   Initialise s
4:   for all step in the episode do
5:     Choose $a$ from $s$ $\epsilon$-greedily using policy derived from $Q$
6:     Take action $a$, observe $r$, $s'$
7:     $Q(s,a) \leftarrow Q(s,a) + \lambda[\gamma \max_{a'} Q(s',a') - (Q(s,a) - r)]$
8:     $s \leftarrow s'$
9:   end for
10: end for
```

In contrast to Q-learning, Sarsa is an on-policy TD algorithm (see Algorithm 4). The temporal difference here is simply the difference between the discounted future prediction and the current Q-value estimate deducted for the observed reward: $\gamma Q(s', a') - (Q(s,a) - r)$ (see line 8 in Algorithm 4). It follows the same policy according to which it estimates Q-values. Thus this algorithm has the property of accumulating the highest reward while optimising that policy. A further modification of this algorithm that includes the estimate of uncertainty of the Q-value for every state-action pair will be given in Chapter 6, along with an explanation of why such an estimate is particularly useful in dialogue management.

In order to optimise the policy, the state-action space must be fully explored, which in real-world problems can be very large. In order to simplify

Algorithm 4 Sarsa

```
1: Initialise $Q(s,a)$ arbitrary for every $(s,a)$ pair
2: for all episode do
3:    Initialise $s$
4:    Choose $a$ from $s$ $\epsilon$-greedily using policy derived from $Q$
5:    for all step in the episode do
6:       Take action $a$, observe $r$, $s'$
7:       Choose $a'$ from $s'$ $\epsilon$-greedily using policy derived from $Q$
8:       $Q(s,a) \leftarrow Q(s,a) + \lambda[\gamma Q(s',a') - (Q(s,a) - r)]$
9:       $s \leftarrow s'$; $a \leftarrow a'$
10:   end for
11: end for
```

the problem, a particular shape of Q-function is usually assumed so that fewer points are needed for it to be approximated. This is often performed by choosing a set of basis functions and defining the unknown Q-function as a parametrised combination of them. These parameters can be then found using gradient methods [53]. This approach be discussed in more detail in Section 2.2.6.

2.2.3 Dialogue as a Markov decision process

Dialogue may be seen as a sequential decision process in terms of the dialogue states, the system actions and the policy. If the dialogue state satisfies the Markov property, then the dialogue can be modelled as a Markov decision process [33, 51, 54, 55, 56, 57, 58, 59]. If an MDP model is to be applied to a real-world dialogue system there are several issues to be resolved. These include how to optimise the policy, what to include in the dialogue state and how to define the reward function.

Modelling dialogue as an MDP allows a policy to be directly optimised via interaction with people. However, many interactions are needed to optimise an MDP policy and hence this approach has only been successful in practice for dialogue systems with a relatively small state space where the system can learn to choose between a small set of actions predefined by an expert for each dialogue state. In [60] this was achieved for a domain which had 149 entities and 3 slots that could each take up to 9 values. However, not all plausible combinations were represented in the dialogue state, so the number of possible dialogue states was small, about 300. There were also only a few possible actions associated with each state that the system could choose from.

Another way of optimising a dialogue policy is via a combination of supervised and reinforcement learning. Supervised learning enables a model of the user to be learned, *i.e.,* transitions between different dialogue states, which is then used to simulate user behaviour. The dialogue manager can then be trained in interaction with the user simulator by means of reinforcement learning. The advantage is that an unlimited number of dialogues can be performed automatically to find the optimal policy [51].

In order to model dialogue as a Markov decision process, the dialogue state must satisfy the Markov property. Methods for defining dialogue state using a corpus of dialogues have been explored in [54]. The basic idea is to define the set of required information at every dialogue step. The dialogue state can then be defined by a small set of boolean indicators for a part of the information that occurs in the corpus. In that way the state space is small enough for policy optimisation to be tractable, but the shortcoming is that its expressibility is greatly restricted. If the dialogue state is significantly extended, it becomes difficult to ensure the tractability of the learning process even if the optimisation takes place in interaction with a simulated user. A way to deal with this problem is to allow the state to incorporate a variety of information, but to optimise the policy in the reduced state space and then perform a heuristic mapping to the original space [55, 61]. Yet another way of ensuring tractability in the policy optimisation is to perform exploration only on the states that are found in a dialogue corpus and to define manually the policy on unexplored parts of the space [57, 62, 63].

The reward function for dialogue policy optimisation can be defined in many different ways. Usually, it contains a measure of how successful the dialogue was, for example whether the information that the user asked for has been given, and how efficient it was, for example how long the dialogue took [51]. It can also contain other parameters such as the number of times the database was queried. The reward function may also depend on the domain – for example, there may be domains where longer dialogues are preferred.

Modelling dialogue as a Markov decision process and adopting and extending the reinforcement learning framework has yielded dialogue policies that outperform hand-crafted rules for dialogue management behaviour according to both objective measures [56] and real user subjective evaluation [58], as well as the human behaviour from a wizard-of-Oz experiment[1] [58].

As an example, consider the NJFun System, which is one of the fist MDP-based spoken dialogue systems built for a real-world task and tested on real users [58]. It provides information about entertainment in New Jersey, covering a large scope of potential user requests. The dialogue state is most compact to enable tractable application of reinforcement learning. It consists of several features that mainly correspond to slots' status flags and also includes ASR confidence scores. The approach taken was to estimate the MDP model from a dialogue corpus and then optimise the policy using model-based reinforcement learning, as discussed in the previous section. The dialogue corpus used for the model estimation was obtained in interaction with real users using a stochastic expert-coded policy. For each dialogue state a set of applicable actions was defined from which the policy then randomly chose. The set of applicable actions was defined in such a way that the stochastic policy still generated reasonable behaviour. Once the model was estimated, the value iteration algorithm (see Algorithm 1) was applied to obtain the optimal policy. The optimal policy was evaluated to real users and compared to both the stochastic policy used for corpus generation and a set of expert-coded policies. In both comparisons the MDP showed an increase in performance.

The Markov decision process offers a model for dialogue management that can make a dialogue manager less dependent on the domain and trainable from data. With enough approximations it can enable real-world spoken dialogue systems to be built. However, a critical requirement for a spoken dialogue system is that it is robust to noise. Some improvements in robustness can be made by taking into account ASR confidence scores in the input [54]. However, the MDP does not include a principled way of dealing with a noisy environment, since it assumes that the dialogue state is fully observable and it therefore does not keep track of alternative dialogue states during the dialogue. In real situations where the user input is corrupted with noise, keeping track of alternative dialogue states can produce a better

[1]This is an experiment where a human replaces a part of the speech application that is being tested while the subjects remain unaware of this [64]. In a spoken dialogue system this is normally the dialogue manager [65].

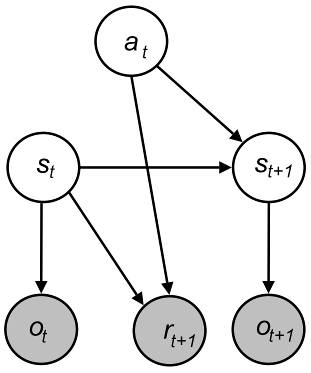


Figure 2.2: Partially observable Markov decision process.

error recovery mechanism. The partially observable Markov decision process (POMDP), discussed in the next section, models the dialogue state as a latent variable, the distribution of which is estimated based on noisy observations. POMDPs therefore offer the possibility of substantially improving the robustness of a spoken dialogue system.

2.2.4 Partially observable Markov decision process

The definition of a partially observable Markov decision process begins with a set of states, $\mathcal{S}$, that a machine can be in, a set of observations, $\mathcal{O}$, that the machine can observe and a set of actions, $\mathcal{A}$, that it can take. Unlike an MDP, the state that the machine is in at a given time s_t is unobservable and must be inferred from observation o_t. The machine takes an action a_t and a transition is made to the next state s_{t+1}. A reward, r_{t+1}, and a new observation, o_{t+1}, are then obtained from the environment. Transitions between the states are defined by a transition probability, $\mathcal{P}^a_{ss'} = P(s_{t+1} = s'|s_t = s, a_t = a)$. The observation probability is $\mathcal{P}_{s'o'} = P(o_{t+1} = o'|s_{t+1} = s')$. The expected reward at every state is given by $\mathcal{R}^a_s = E(r_{t+1}|s_t = s, a_t = a)$.

A partially observable Markov decision process can be represented as a dynamic Bayesian network, a part of which is given in Fig. 2.2, where shaded nodes denote observable variables. In contrast to Fig. 2.1 the states are hidden and must be inferred from observation. Similar to the MDP, a geometric discount factor $\gamma \in [0, 1]$ is used to place more importance on the rewards obtained early in the process.

At each time step, the machine is in a certain state s_t, but since the state is unobservable the machine maintains a distribution over all possible states at time t – called the *belief state*, $b(s_t)$. Then, the probability of the machine being in state s, $s \in \mathcal{S}$, at time t is $b(s_t = s)$. Belief state $b(s_t)$ therefore takes values $\mathbf{b} \in \mathcal{B}$ where $\mathcal{B} = [0,1]^{|\mathcal{S}|}$ is the *belief space*. The initial distribution over states $b(s_0)$ is given by $\mathbf{b}^0 = [b(s_0 = s^1), \ldots, b(s_0 = s^{|\mathcal{S}|})]^\mathsf{T}$. The initial distribution $\mathbf{b}^0$, together with the observation probabilities $\mathcal{P}_{s'o'}$, the transition probabilities $\mathcal{P}^a_{ss'}$ and the reward expectations $\mathcal{R}^a_s$, fully specifies the POMDP model.

When a new observation arrives, the new belief state $b(s_{t+1})$ is obtained as the probability distribution over all possible states at time $t+1$, given the observation o_{t+1}, the action taken a_t, and the previous belief state $b(s_t)$. Provided that the state space is finite, the new belief state can be computed as follows:

$$\begin{aligned}
& b(s_{t+1} = s') \\
& = P(s_{t+1} = s' | o_{t+1} = o', a_t = a, b(s_t) = \mathbf{b}) \\
& = \frac{P(o_{t+1} = o' | s_{t+1} = s', a_t = a, b(s_t) = \mathbf{b}) P(s_{t+1} = s' | a_t = a, b(s_t) = \mathbf{b})}{P(o_{t+1} = o' | a_t = a, b(s_t) = \mathbf{b})} \\
& \propto P(o_{t+1} = o' | s_{t+1} = s') \\
& \quad \cdot \sum_{s \in \mathcal{S}} P(s_{t+1} = s' | a_t = a, b(s_t) = \mathbf{b}, s_t = s) P(s_t = s | a_t = a, b(s_t) = \mathbf{b}) \\
& = \underbrace{P(o_{t+1} = o' | s_{t+1} = s')}_{\mathcal{P}_{s'o'}} \sum_{s \in \mathcal{S}} \underbrace{P(s_{t+1} = s' | a_t = a, s_t = s)}_{\mathcal{P}^a_{ss'}} b(s_t = s), \qquad (2.10)
\end{aligned}$$

for every $s' \in \mathcal{S}$, where the belief state at time t, $b(s_t)$, is given by $\mathbf{b} = [b(s_t = s^1), \ldots, b(s_t = s^{|\mathcal{S}|})]^\mathsf{T}$, action a is taken at time t and observation o' is observed at time $t+1$.

2.2.5 POMDP policy optimisation

Since the state in the POMDP always remains unobservable, the policy in the POMDP is a mapping between belief states and actions. Unfortunately, the continuous nature of the belief states makes policy optimisation in a POMDP model non-trivial and computationally expensive. An outline of the Value iteration algorithm for POMDPs is now given, based on a more detailed description in [34]. In order to apply the value iteration algorithm the state space, the action space and the observation space must be discrete and finite.

A POMDP policy π provides an action a from the action space $\mathcal{A}$ for every element $\mathbf{b}$ of the belief space $\mathcal{B}$, $\pi : \mathcal{B} \to \mathcal{A}$. Such a policy does not change with time and is therefore stationary. If the POMDP model does not change over time, one can show that there is an optimal policy which is also stationary [34].

The optimal Value function in each state is, similarly to the Bellman equation for MDPs (Eq. 2.5):

$$V(s) = \max_a \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \cdot \left(\mathcal{R}^a_s + \sum_{o' \in \mathcal{O}} \mathcal{P}_{s'o'} \gamma V(s') \right). \tag{2.11}$$

In the same way, the optimal Q-function in each state and action pair is:

$$Q(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \cdot \left(\mathcal{R}^a_s + \max_{a'} \sum_{o' \in \mathcal{O}} \mathcal{P}_{s'o'} \gamma Q(s',a') \right). \tag{2.12}$$

Eqs. 2.11 and 2.12 yield the optimal Value function in every belief state, $V : \mathcal{B} \to \mathbb{R}$, as the expectation of the optimal Value function over all states,

$$V(\mathbf{b}) = \sum_{s \in \mathcal{S}} b(s_t = s) V(s), \tag{2.13}$$

and the optimal Q-function in every belief state and action pair, $Q : \mathcal{B} \times \mathcal{A} \to \mathbb{R}$, as the expectation of the optimal Q-function over all states,

$$Q(\mathbf{b}, a) = \sum_{s \in \mathcal{S}} b(s_t = s) Q(s,a), \tag{2.14}$$

where $\mathbf{b}$ is the vector of probabilities $[b(s_t = s^1), \ldots, b(s_t = s^{|\mathcal{S}|})]^\mathsf{T}$.

As with MDP value iteration, if the optimal policy is estimated in an iterative process, it is non-stationary during the process of optimisation but it converges to a stationary policy (Section 2.2.2).

A non-stationary POMDP policy can be seen as a function of the observation and the number of time steps needed to reach the final step. If the machine is one time step before reaching the final state, then it obtains one observation, $o \in \mathcal{O}$, it takes an action specified by the policy for that time step and observation o. In t time steps to the final state (t-step) it

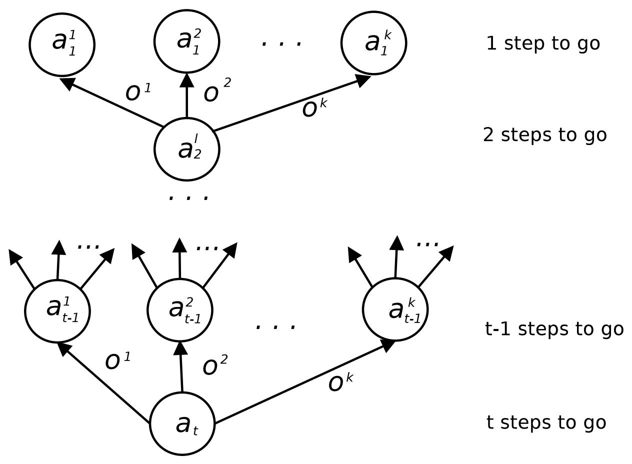


Figure 2.3: A t-step policy tree.

obtains an observation, takes an action specified by the policy and then follows the $(t-1)$-step policy – the policy defined for $t-1$ time steps. Thus, the (non-stationary) policy may be viewed as a tree of depth t which has an action associated with each node and each node has $|\mathcal{O}|$ edges – one for each observation. This tree is called the *policy tree*, see Fig. 2.3, where $\mathcal{O} = \{o^1, \ldots, o^k\}$ and $a_i^j \in \mathcal{A}$, $i \in \{1, \ldots, t\}$ and $j \in \{1, \ldots, k\}$.

Eq. 2.11 allows a relation between the Value functions at two time steps to be established:

$$V^{\pi}(s) = \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{a(\pi)} \left(\mathcal{R}_s^{a(\pi)} + \sum_{o' \in \mathcal{O}} \mathcal{P}_{s'o'} \gamma V^{o'(\pi)}(s') \right), \tag{2.15}$$

where $a(\pi)$ is the action at the root of policy tree π, $V^{\pi}(s)$ is the Value function of policy tree π in state s and $V^{o'(\pi)}(s')$ is the Value function in state s' for the upper part of policy tree π where o' is observed. In this way, $V^{\pi}(s)$ relates to a t-step policy tree and $V^{o'(\pi)}(s')$ to a $(t-1)$-step policy tree.

The Value function for the t-step policy tree π in the belief state at time t is the expectation of the Value function for policy tree π over all states,

$$V^{\pi}(\mathbf{b}) = \sum_{s \in \mathcal{S}} b(s_t = s) V^{\pi}(s), \tag{2.16}$$

where $\mathbf{b}$ is the vector of probabilities $[b(s_t = s^1), \ldots, b(s_t = s^{|\mathcal{S}|})]^{\mathsf{T}}$.

If Π_t is the set of all policy trees at step t, the t-step optimal Value function is the Value function of the best policy tree at step t:

$$V_t(\mathbf{b}) = \max_{\pi \in \Pi_t} \sum_{s \in \mathcal{S}} b(s_t = s) V^{\pi}(s). \tag{2.17}$$

From Eq. 2.16, it follows that each policy tree at time t has a Value function which is linear in the belief state. The optimal value at time step t is the upper surface of these functions yielded by all possible policy trees at time t, see Eq. 2.17. Therefore, the Value function for a POMDP is piecewise linear and convex for any finite t.

Consider, for example, a simple POMDP with state space $\mathcal{S} = \{s^1, s^2\}$ and action space $\mathcal{A} = \{a^1, a^2, a^3\}$, and some finite observation set and the rewards are $r(s^1, a^1) = 1$, $r(s^1, a^2) = 4.5$, $r(s^1, a^3) = 4$, $r(s^2, a^1) = 5.5$, $r(s^2, a^2) = 2$ and $r(s^2, a^3) = 4$. The value iteration algorithm starts at the 1-step, where all policy trees contain only one node, since there is only one step to go. Therefore, there are three different policy trees, one for each action from $\mathcal{A}$. In Fig. 2.4, the Value function of each policy tree is given. The horizontal axis represents the belief state. Since there are only two states, the belief in one state determines the belief in the other, $b(s^1) = 1 - b(s^2)$. Thus the belief state can be represented in one dimension. The vertical axis represents the expected discounted reward for 1 step – the 1-step Value function. In the 1-step of value iteration, the interpretation of the expected discounted reward is just the reward that is obtained after taking one action. When $b(s^1)$ is equal to 0 the machine is certain to be in state s^2 so the Value function is the reward that is obtained by taking a particular action in state s^2. The case when the machine is in state s^1 is similar. Since the Value function is linear in $b(s_t)$, the Value function for each 1-step policy tree is fully specified by its value at these two end-points. In general, the Value function of each policy tree is a multidimensional hyperplane where the number of dimensions is $|\mathcal{S}| - 1$. The upper surface of these Value functions represents the optimal 1-step Value function, which, in the example in Fig. 2.4, is denoted by a full line, showing that the POMDP Value function is piecewise linear and convex.

Each policy tree can represent the optimal strategy at some point in the belief space and therefore it can contribute to the final optimal Value function. However, if the t-step Value function of policy π is under the t-

step optimal Value function, the policy π does not contribute to the (overall) optimal Value function. Therefore, such a policy tree does not have to be taken into account for the next step of the value iteration algorithm. Each iteration of the value iteration algorithm for policy optimisation thus consists of two parts: a generation step – updating the Value function at step $t+1$ using the Value functions of the policy trees in step t, and a pruning step – removing the policy trees in which the Value function fall under the surface of the optimal Value function in step $t+1$. This process is repeated until there is no change in the optimal Value function estimation for two consecutive steps.

Given the set of the Value functions of pruned policy trees at time step $t-1$, $\mathcal{V}^+_{t-1}$, the set of actions $\mathcal{A}$ and a set of observations $\mathcal{O}$, the number of operations needed to obtain the set of the Value functions of policy trees at time step t, $\mathcal{V}_t$ is $|\mathcal{A}||\mathcal{V}^+_{t-1}|^{|\mathcal{O}|}$. Thus, every iteration of the value iteration has exponential complexity which makes it intractable even for small scale problems. Although there are ways of making the algorithm more efficient, such as the Witness algorithm [34] or using an approximate solution, as in the Point-based value iteration algorithm [66, 67], these are only suitable for relatively small action/state problems and thus not directly applicable for large state space applications. This is particularly the case for dialogue management in real-world problems where the observation set can be infinite.

An alternative policy optimisation process is based on so-called *grid-based learning*. On the assumption that the POMDP state $\mathcal{S}$ is discrete and finite and that the belief state can be tractably maintained, the POMDP model can be viewed as an MDP model in which the state space is the belief space of the original POMDP, which is $[0,1]^{|\mathcal{S}|}$ [34]. The transition model of that MDP is defined by the belief update formula Eq. 2.10. If a distance metric is defined on $[0,1]^{|\mathcal{S}|}$ it can be used to define a finite number of points in this space, *grid points*, such that the distance between any two points from this space is not smaller than a certain threshold ν and that for every non-grid point in this space there exists a grid point such that the distance between the two is less then ν. Every time a point in the belief space is visited it is approximated with its closest grid point. The set of grid points then define a discrete MDP. This allows for (discrete) MDP algorithms to be used to find an optimal policy on the grid points [68]. However, the tran-

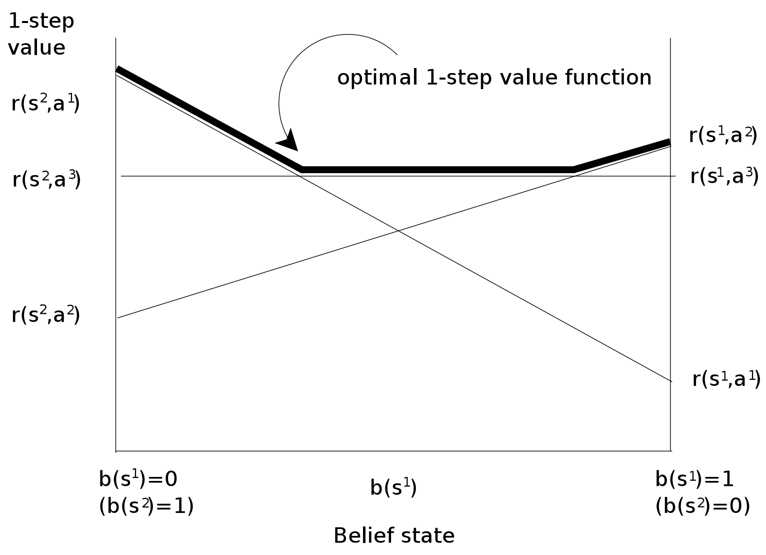


Figure 2.4: The optimal 1-step Value function.

sition model of this MDP is unknown, since many belief states map to the same grid point. While the value iteration algorithm guarantees convergence to the true policy given enough iterations, it requires the transition model to be known. The speed of convergence not only depends on the number of iterations, but also on the number of grid-points. If the number of grid points is too small, then approximating a belief state point with its closest grid point can lead to suboptimal results since two very different belief state points can be approximated with same grid point. Conversely, if the number of grid-points tends to infinity, the MDP policy optimisation becomes intractable.

2.2.6 Dialogue as a partially observable Markov decision process

The partially observable Markov decision process framework offers a principled way of handling the intrinsic uncertainty that occurs in dialogue by assuming only partial observability of the dialogue state [20, 21, 69, 70, 71, 72, 73, 74].

When a dialogue is modelled as a partially observable Markov decision process, the dialogue manager maintains a distribution over states at each dialogue step – the belief state. The policy then maps the belief state to an appropriate action. Thus the uncertainty is taken into account in the policy optimisation process and the system can learn how to behave optimally under differing levels of uncertainty to achieve a successful dialogue.

For simple domains, when the number of states is around 10, one can tractably maintain the exact belief state and use approximate policy optimisation. Past researchers have shown that the performance of such a dialogue manager outperforms the equivalent MDP [21, 69]. However, problems occur when applying this approach to larger domains, particularly when the number of states is significantly larger. Two of the main issues that arise are effectively maintaining the belief state and tractably performing policy optimisation.

In order to deal with these limitations, one can factor the dialogue state to include some reasonable conditional independence assumptions that apply in many situations and still satisfy the Markov property [75]. To explain this it is useful to reproduce the belief update formula Eq. 2.10:

$$b(s_{t+1} = s') \propto P(o_{t+1} = o'|s_{t+1} = s') \sum_{s \in \mathcal{S}} P(s_{t+1} = s'|a_t^m = a_m, s_t = s) b(s_t = s), \tag{2.18}$$

where s_t is the dialogue state at time t, a_t^m is the system's action[1] at time t, o_{t+1} is the observation at next time step. The dialogue state s_t can be composed of the user goal s_t^u, the user action a_t^u and the dialogue history s_t^d. The belief state then becomes a joint probability distribution of the user goal, the user action and the dialogue history, $b(s_t) = P(s_t^u, a_t^u, s_t^d)$. This will be denoted as $b(s_t^u, a_t^u, s_t^d)$. The user goal represents the task that the system has to fulfil – for example, provide information about an entity with particular attributes. The user action represents the set of semantic concepts that the user intended to communicate to the system. The dialogue history records those events that are important to maintaining the dialogue flow – for example when the user made a confirmation or the system requested information. Several conditional independence assumptions are made. Firstly, the observation depends solely on the (true) user action, *i.e.,* what the user intended to say. Secondly, the user action depends only on the system's action and the user goal. Thirdly, the user goal depends

[1]In dialogue, both user and system take actions. In order clearly to distinguish between the two, the system's action at time t is denoted as a_t^m and its instance as a_m.

only on the previous user goal and the system's action. Finally, the dialogue history depends only on the previous dialogue history, the user goal, the user action and the system's action. These assumptions can be represented as the dynamic Bayesian network, as shown in Fig. 2.5.

This factorisation makes it possible to simplify the belief update process. This process now yields four update components: the observation model, the user action model, the user goal model and the dialogue history model, see Eq. 2.19 where $\mathcal{S}^u$ is a set of all user goals, $\mathcal{A}^u$ is a set of all user actions and $\mathcal{S}^d$ is a set of all dialogue histories.

$$\begin{aligned}
&b(s^u_{t+1} = s'_u, a^u_{t+1} = a'_u, s^d_{t+1} = s'_d) \\
&\propto P(o_{t+1} = o' | s^u_{t+1} = s'_u, a^u_{t+1} = a'_u, s^d_{t+1} = s'_d) \\
&\quad \sum_{s_u \in \mathcal{S}^u, a_u \in \mathcal{A}^u, s_d \in \mathcal{S}^d} P(s^u_{t+1} = s'_u, a^u_{t+1} = a'_u, s^d_{t+1} = s'_d | a^m_t = a_m, s^u_t = s_u, a^u_t = a_u, s^d_t = s_d) \\
&\qquad b(s^u_t = s_u, a^u_t = a_u, s^d_t = s_d) \\
&= P(o_{t+1} = o' | a^u_{t+1} = a'_u) \\
&\quad P(a^u_{t+1} = a'_u | s^u_{t+1} = s'_u, a^m_t = a_m) \\
&\quad \sum_{s_u \in \mathcal{S}^u, a_u \in \mathcal{A}^u, s_d \in \mathcal{S}^d} P(s^u_{t+1} = s'_u | s^u_t = s_u, a^m_t = a_m) \\
&\qquad \cdot P(a^u_{t+1} = a'_u | s^u_{t+1} = s'_u, a^m_t = a_m) \\
&\qquad \cdot P(s^d_{t+1} = s'_d | s^u_{t+1} = s'_u, a^u_{t+1} = a'_u, s^d_t = s'_d, a^m_t = a_m) b(s^u_t = s_u, a^u_t = a_u, s^d_t = s_d) \\
&= \underbrace{P(o_{t+1} = o' | a^u_{t+1} = a'_u)}_{\text{observation model}} \\
&\quad \underbrace{P(a^u_{t+1} = a'_u | s^u_{t+1} = s'_u, a^m_t = a_m)}_{\text{user action model}} \\
&\quad \sum_{s^u \in \mathcal{S}^u} \underbrace{P(s^u_{t+1} = s'_u | s^u_t = s_u, a^m_t = a_m)}_{\text{user goal model}} \\
&\qquad \sum_{s^d \in \mathcal{S}^d} \underbrace{P(s^d_{t+1} = s'_d | s^u_{t+1} = s'_u, a^u_{t+1} = a'_u, s^d_t = s_d, a^m_t = a_m)}_{\text{dialogue history model}} \\
&\qquad\quad \sum_{a^u \in \mathcal{A}^u} b(s^u_t = s_u, a^u_t = a_u, s^d_t = s_d)
\end{aligned} \tag{2.19}$$

Using this factorisation proved to be a highly effective way of maintaining the belief for state spaces of order 1000 [75]. However, applying this to

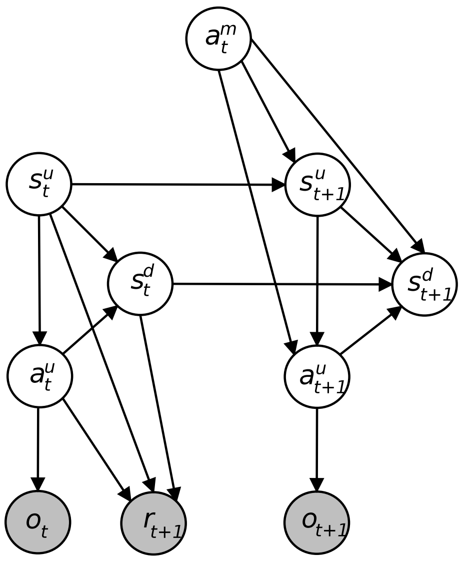


Figure 2.5: Partially observable Markov decision process for dialogue modelling.

even larger domains requires more approximations. One way of achieving tractability in belief maintaining in a real-world domain is to group states together and thus maintain the belief only on the most probable states [76] (a more detailed description will be given in Chapter 3).

An alternative way of achieving tractability is to factor the state space further into independent elements that are derived using the expert knowledge about the domain. This approach has been taken in [71], where the dialogue state is factorised into fully independent elements. This then allows for the parametrised transition and observation probabilities to be assumed for each state element and the belief state to be updated on each of these elements independently, using belief propagation. This idea was further developed in the Bayesian Update of Dialogue State (BUDS) system [77], where only conditional independence between elements of the state is assumed and the belief is then updated using loopy belief propagation. An alternative approach makes use of particle filters to update the belief [78]. However, this approach exhibited a response time of 3 seconds for a domain with 20 slots where each slot had up to 13 values. On the other hand, the approach taken in BUDS was able to update belief in real time for a domain with 42 entities and 10 slots [77] and a domain with 16 slots some of which had more than $300,000$ values [79]. It was also shown in [73] that the parameters of the transition and observation probabilities can be tied and learnt automatically from a dialogue corpus using expectation propagation. However, the problem remains of how efficiently to maintain belief state for large dialogue domains for dialogues of unlimited length if no independence assumptions are considered among the elements in dialogue state. These issues will be addressed in this thesis within the Hidden Information State framework in Chapter 4.

The second cause of intractability is policy optimisation. Dialogue policy optimisation using POMDP learning methods was explored in [21, 69, 71, 80], and has been shown to yield better performance than MDPs. However, the state space of these systems is still modest relative to real-world applications. Alternative solutions view the POMDP as a continuous-space MDP and train the policy in interaction with a simulated user. One approach is to apply grid-based methods to discretise the space and then use MDP policy optimisation algorithms. Such an approach was explored in [76] for a real-world dialogue task and will be explained in more detail in Section 3.3.5. Another approach, taken in the BUDS system [77], is to use a parametric approximation of the optimal policy. The following steps are taken. A set of feature functions is defined on the full belief space using expert-knowledge of the domain. A grid-based method is then applied to create feature vectors with discrete values, which are finally used for policy approximation. This allows gradient methods to be applied to find the optimal policy parameter values (about 800 parameters). The optimal policy is then found using Natural Actor Critic [53] algorithm. This is a reinforcement learning algorithm which makes use of the natural gradient [81]. The reason it performs well is that the natural gradient makes use of the shape of the space to find the steepest decent, and thus is highly efficient [77].

On-line learning in direct interaction with real users is of a particular interest in the area of spoken dialogue systems since it allows the optimisation and adaptation of the policy while it is used. This thesis will address the problem of policy optimisation using a Bayesian non-parametric approach for function approximation, with the aim of making the optimisation, on the one hand, less dependent on the expert-knowledge and, on the other hand, faster so that it can be learned or improved in direct interaction with real people (Chapters 6 and 7).

2.3 User simulator

When adopting a reinforcement learning approach to dialogue policy optimisation, either model-based or model-free optimisation algorithms can be applied (see Section 2.2). In order to apply the model-based approach to an MDP model, state transition probabilities as defined in Section 2.2.1 need to be available. For an POMDP model both transition and observation probabilities need to be known (Section 2.2.4). Given an annotated dialogue corpus these probabilities can be estimated. Such an approach faces sparsity problems and requires the use of further approximations [82, 83]. As a result, the model-free approach, where the policy can be optimised directly in interaction with users, is often more appropriate. Although it is theoretically possible to optimise a policy in direct interaction with real users, for example via large call centres which serve tens of thousands of calls per day, the policy's performance at the initial stage of learning is typically too low to be considered acceptable. Instead, it is common practice in a reinforcement learning approach to dialogue management that a simulated user is used to interact with the dialogue manager during policy optimisation [51, 77, 84, 85, 86, 87, 88].

There are many techniques that can be used for building a simulator, for example graph-based [89] and agenda-based [90] techniques. These simulators usually aim to exhibit two key characteristics: reasonable, goal-directed behaviour and variability in the way it interacts with the dialogue manager so that it can represent a wide coverage of potential users. The agenda-based user simulator described in [90] was used in this thesis and a more detailed description of its operation and evaluation will be given in Section 3.4.2.

There are, however, various statistical methods for learning the user simulator behaviour from a dialogue corpus. This is in principle more desirable since that makes the process of dialogue modelling fully automatic [51]. Examples include n-gram methods [91], a simulated user built using Bayesian networks [92], cluster-based user simulation [87], hidden-agenda [93] and agenda-based user simulation with parameters estimated from data [94].

2.4 Evaluation

While methods for evaluation of most data-driven tasks in speech and natural language processing are well-established [95], evaluating a spoken dialogue system requires interaction and is therefore difficult [96]. A spoken dialogue system is built of distinct modules and although for most of them there are defined methods of evaluation, joint evaluation of the complete system is challenging. Evaluating the dialogue manager's performance is hard in itself, because of the vast space of possible dialogues.

The usual approach to evaluation is to have the dialogue manager interact with humans, and let human judges rate the dialogues [38, 58]. This makes it infeasible to evaluate all possible dialogues and is also costly and time-consuming to perform. An alternative evaluation metric is to use the reinforcement learning reward function [51], replacing the human judges.

Letting the user simulator instead of human users interact with the dialogue manager enables a wider coverage of dialogue space in evaluation. A simulated user can generate an unlimited number of dialogues with a variety of scenarios and varying noise levels [10, 77, 90, 97, 98]. A disadvantage of this approach is the potential discrepancy between real user and a simulated user behaviour.

This work will use a simple reward function effectively used in other studies [10, 77, 90] which encourages short and successful dialogues. An alternative would be to derive a reward function from ratings by human judges. The PARADISE evaluation framework [99] predicts the user satisfaction ratings using a corpus of dialogues annotated with real user satisfactions and a set of objective measures. It defines the dialogue performance as a weighted function of the dialogue success and dialogue-based cost measures, such as the dialogue length or the number of times the system produced a confirmation. The weights can be inferred from the annotated corpus using regression. The measures are normalised so that they are applicable across domains. While such a complex reward function may be desirable, for the purposes of this thesis the use of a simple reward function is sufficient.

The main evaluation approach that will be taken in this thesis is to generate dialogues with the agenda-based simulated user in a range of noise levels and to evaluate them using the reward function. Section 3.4 will give details on these. This thesis will also use more fine-grained evaluation metrics. These are the success rate and the average length of the dialogue, the constituent parts of the reward function. In addition, the notion of success may be further divided into partial and full completion of the underlying task that the dialogue manager is supposed to fulfil.

It is important to note that evaluations on the simulated user will be used here to contrast different methods that themselves do not encode any user-dependency. Therefore conclusions drawn from these comparisons should hold even if the system was trained and tested using a different user simulator. The aim of these contrasts is to evaluate effectiveness of a particular technique inside the dialogue manager rather than the dialogue manager as a whole.

2.5 Summary

This chapter has outlined the research carried out in the domain of statistical spoken dialogue systems for task-oriented dialogues. Statistical approaches to dialogue modelling are typically based on the Markov decision process model. The MDP model assumes that the dialogue can be viewed in terms of the dialogue state, the system's action and the reward. Reinforcement learning algorithms then allow the optimal dialogue policy to be obtained. This chapter has offered a review of several reinforcement learning algorithms that are most commonly used to solve MDPs. In order to deal with the intrinsic uncertainty in the dialogue state that occurs due to speech understanding errors the dialogue can also be modelled as a partially observable Markov decision process. This chapter has reviewed the basic algorithms used to obtain the optimal policy in a POMDP and identified the problem of tractability as the main obstacle to applying reinforcement learning methods to real-world problems.

The next chapter will provide a detailed explanation of the Hidden Information State model, which will be used for the rest of this thesis.

Chapter 3

A real-world POMDP-based dialogue manager

3.1 Introduction

The Hidden Information State (HIS) [76] system was one of the first scalable POMDP-based dialogue systems able to sustain conversation in real time and be tested on real users. It is based on an end-to-end statistical approach to building spoken dialogue systems [20] and incorporates a POMDP model for dialogue management in such a way that dialogues from a large domain are made possible [35, 76, 100]. In this chapter an overview of the complete system is given (Section 3.2) followed by an in-depth description of the dialogue manager component. Section 3.4 then explains how training and evaluation are performed in interaction with a simulated user. This is followed with a description of a real user evaluation (Section 3.5). Section 3.6 concludes the chapter.

3.2 The Hidden Information State system

The first prototype of the HIS system was built to provide tourist information for tourists in the fictitious town of Jasonville. The system uses a database of 42 entities such as restaurants, hotels and bars. Each of them have a set of up to 10 attributes that the user can query. Using natural language, the user can seek an entity with particular constraints and then obtain information about each entity mentioned by the system.

3.2.1 System structure

The HIS system is structured as a collection of modules, as was shown in Fig. 1.1. Speech recognition uses the ATK/HTK toolkits [101] with the acoustic model trained on 40 hours of in-domain speech data, and the language model trained on 80M words. 55K of these words were in-domain and interpolation was used to incorporate them into the full language model. The recogniser has a vocabulary of 2000 words. An important feature of the recogniser is that it produces a scored N-best list of possible user utterances for each utterance. In the ATK recogniser, the confidence score of a hypothesised utterance is computed as a product of the posterior probabilities of corresponding word arcs in the confusion network. When normalised, this gives an estimate of the probability of each hypothesised utterance [102]. This input structure is used as an observation model in the belief update process (see Eq. 2.19 in Section 2.2.6). The accuracy of these probabilities are therefore particularly important if the dialogue manager is to be robust to speech recognition errors.

The N-best list of hypothesised utterances is passed from the speech recogniser to the semantic decoder, which decodes every utterance separately to form a semantic representation of the user input, called a *dialogue act*. A statistical semantic decoder is used, trained using support vector machine (SVM) classifiers to provide the semantic concepts for each utterance in the N-best list [30]. Since several recognition hypotheses may map to the same dialogue act, the confidence score of each act is calculated as the sum of the confidence scores of all utterances that map to that dialogue act.

The output from the dialogue manager is also in the form of a dialogue act. It is passed to the natural language generator which converts it to a natural language sentence using simple, template-based rules. Finally the speech is synthesised using an HMM speech synthesiser with probabilistic modelling of F0 and using a globally tied distribution in unvoiced regions [103].

An example dialogue with a real user in the TownInfo domain is given in Table 3.3. It starts with an opening question from the system that allows the user to take initiative and specify their request. For the initial turn the internal output of each of the system's modules is provided. The speech recogniser provides a 10-best list of possible user utterances for a given

System:	Hello, how may I help you?
User:	Hi. Could I have the name of a Russian restaurant which has 3 stars please?
ASR	I can I have the name of a Russian restaurant which has three stars please.
	I can I have the name of a Russian restaurant which jazz three stars please.
	I can I have the name of a Russian restaurant which last three stars please.
	I can I have the name of a Russian restaurant which has three stars please.
	I I I have the name of a Russian restaurant which has three stars please.
	I can I have the name of the Russian restaurant which jazz three stars please.
	I can have the name of a Russian restaurant which has three stars please.
	I I I have the name of a Russian restaurant which jazz three stars please.
	I can I have the name of the Russian restaurant which last three stars please
	I can have the name of a Russian restaurant which jazz three stars please.
SD	confirm(food=Russian,type=restaurant,stars=3) 0.50
	confirm(food=Russian,type=restaurant,music=jazz, stars=3) 0.24
	inform(food=Russian,type=restaurant,stars=3) 0.16
	inform(food=Russian,type=restaurant,music=jazz,stars=3) 0.10
DM	inform(name="Saint Petersburg",type=restaurant,food=Russian, stars=3, music=ethnic)
System:	Saint Petersburg is a nice restaurant, they serve Russian food, it has 3 stars, they play ethnic music.
User:	What's the price range please?
System:	The Saint Petersburg is in the moderate price range.
User:	What's the address and phone number please?
System:	Sure, the Saint Petersburg is on Park Road.
User:	What's the telephone number please?
System:	Sure the Saint Petersburg is on Park Road; and the phone number is 702-70-03.

Table 3.1: A TownInfo dialogue.

speech signal[1]. This 10-best list is passed to the semantic decoder (SD) which then generates a 4-best list of dialogue acts with associated confidence scores from this input. One can see how the prominent confusion between jazz and has in the ASR 10-best list has resulted in the concept music=jazz occurring in the semantic decoder's 4-best list. The dialogue manager (DM) then decides on a system action based on this list and its beliefs. The action-selection process as well as the internal dialogue manager's representation of the dialogue state will be explained in more detail in the following section.

3.3 The Hidden Information State dialogue manager

The Hidden Information State dialogue manager is a POMDP-based dialogue manager that can maintain its belief state during dialogues in a large domain. Its belief update is based on a further approximation of the dialogue state factorisation described in [75].

3.3.1 Dialogue act formalism

As far as dialogue management is concerned, the dialogue between the system and the user takes place at dialogue act level. A dialogue act is an internal representation of the user intention conveyed in the user utterance. The dialogue act representation used here assume the act consists of a sequence of semantic concepts. The semantic concepts should be compact enough that their number is kept relatively low. On the other hand, they should carry enough information to sustain the dialogue flow. The dialogue act formalism incorporates expert knowledge about the domain. However, it is based on a well-established theory [104] and as such is extendible to other task-driven dialogue domains.

A dialogue act in the HIS system consists of the dialogue act type and a list of attribute-value pairs. The dialogue act type defines the intention conveyed in the utterance. For example, if the user is passing some information to the system regarding their request, that corresponds to the inform dialogue act type. If the user or the system is confirming a particular attribute-value pair, then a confirm dialogue act type is used. The list of attribute-value

[1]For simplicity the ASR confidence scores are omitted.

pairs refers to the specific concepts mentioned by the user or the system. For instance, the utterance *"Is that restaurant in the centre?"* would be represented by the dialogue act confirm(area=central, type=restaurant). In some cases the order of the attribute-value pairs in the list is important. For example, if the user said *"I don't want riverside area, I want something in the centre."*, the corresponding dialogue act would be deny(area=riverside, area=central). A complete list of dialogue acts with their descriptions is given in Appendix A.

3.3.2 Domain ontology

As already noted in Section 2.1.2, dialogue managers normally keep a fixed list of slots that are filled in during the dialogue. The HIS dialogue manager uses a more structured representation via an *ontology* which defines the relationship between different attributes and values that can occur in the dialogue act. The ontology has a tree structure. The tree nodes are divided into three groups: class nodes, lexical nodes and atomic nodes (see Fig. 3.1). Class nodes can have many child nodes. The first is always atomic and defines a specific instance of the class. The remainder consist of an optional class node and one or more lexical nodes. Lexical nodes can have only a single atomic child node.

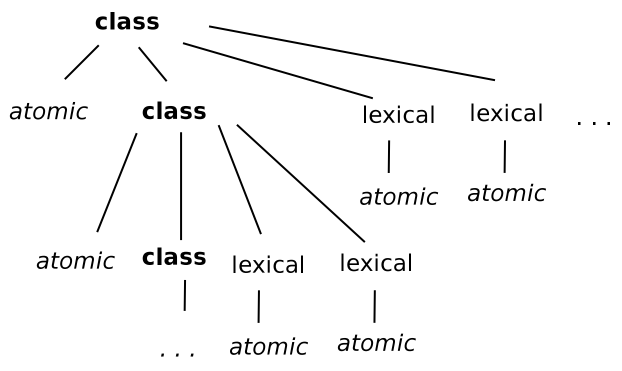


Figure 3.1: Generic ontology structure.

The TownInfo ontology is given in Table 3.2, where examples of class nodes are **entity** and **type** (bold font), lexical nodes are pricerange and food (regular font) and atomic nodes are *restaurant* and *Chinese* (italic font).

The attributes listed in each dialogue act correspond to either class or lexical nodes in the ontology; and the values that they take are represented by atomic nodes. The tree root is a class node and it defines the user goal in the most general way. Other class nodes define the user query in a more precise manner. More specifically, each class node and its atomic child node define an additional set of attributes that are represented by lexical nodes and optionally a class node. For example, in the ontology from Table 3.2, the atomic node restaurant for class node type defines an additional set of lexical nodes: food, pricerange, music, drinks, and stars.

entity	$\leftarrow$	*venue*(**type**, area, name, address, near, phone, comment)
type	$\leftarrow$	*hotel*(pricerange, stars, price, drinks)
type	$\leftarrow$	*restaurant* (food, pricerange, price, music, drinks, stars)
type	$\leftarrow$	*bar*(drinks, music, pricerange)
type	$\leftarrow$	*amenity*
area	=	{ *central*, *east*, *west*, ... }
food	=	{ *Italian*, *Chinese*, *Indian*, ... }
pricerange	=	{ *cheap*, *expensive*, ... }

Table 3.2: TownInfo ontology rules.

This allows the dialogue manager to make use of the hierarchical relationship between the attributes to model the dependencies in each user input. For example, in the tourist information domain if the user specified food=Italian, that would imply that the user wants type=restaurant and entity=venue. Attribute-value pair pricerange=cheap is associated with type=restaurant, type=bar and type=hotel, but not type=amenity.

3.3.3 State representation

One of the main obstacles in implementing a POMDP-based dialogue manager in real-world systems is the intractability of the belief update equation (Eq. 2.19 in Section 2.2.6). Adopting the factored state formalism [75], each dialogue state consists of the user goal, the dialogue history and the last user act. In any real-world dialogue, this combination can result in a vast number of dialogue states and it would not be computationally tractable to maintain a probability distribution over such a large state space.

As already noted in Section 2.2.6, the user goal represents a task that the system has to fulfil. In this domain, the task is to give information about an entity which has attributes with certain values. The set of all possible user goals is thus a set of all plausible combinations of attribute-value pairs, which is determined by the domain ontology. It is clearly intractable to maintain a probability distribution over such a large set. Therefore, in the HIS system, the idea is to group user goals together into *partitions* and maintain the probability over partitions rather than over all the user goals. The set of partitions is such that any user goal is either a partition, or there is a partition that that user goal belongs to. The set of partitions is created dynamically during the dialogue. The partitions are split recursively using the attribute-value pairs from the user input and the system output.

There are two main requirements for building partitions. First, one must ensure that every partition is unique. Second, the partition representation should be compact enough that the system can efficiently store them. As already mentioned, the partitions are split using the attribute-value pairs from the N-best list of the user input and the system output. They are combined together using the relationships defined by the domain ontology, see Table 3.2. This results in each partition being a tree from the ontology with a single value for each partition node. In the original implementation of the HIS system, nodes that have the same child nodes in different partitions are shared among the partitions they appear in. This reduces the memory requirement for storing the partitions.

An example of the partitioning process is given in Fig. 3.2. Initially there is only one generic partition entity denoted by a box in Fig. 3.2. This partition corresponds to the root of the ontology tree. The system asks the opening question *"Hello, how may I help you?"* Then, the user input *"I'm looking for an expensive Italian restaurant in the centre, please."* is received and semantically decoded as inform(type=restaurant, pricerange=cheap, food=Italian, area=central). Each attribute-value pair is recursively used for splitting the generic partition. The partition splitting process is the following. Attribute-value pair type=restaurant applies to the ontology tree where entity is venue. Therefore, class node entity is split into node entity in such way that the node entity is replicated and has a child atomic node venue. In that way, the generic partition is split into two partitions – one which has nodes entity and venue and other which just has the node entity. Since they appear in both partitions, nodes entity are connected in an array. Node venue now determines other child nodes of node entity in the partition it belongs to.

These are defined by the ontology. For simplicity, only type and area are shown in Fig. 3.2 and connected to node entity. Next, applying attribute-value pair type=restaurant splits node type by replicating it as node type with child atomic node restaurant. Similar to atomic node venue, node restaurant also defines other child nodes of node type and one of these nodes is lexical node pricerange. Attribute-value pairs pricerange=cheap, food=Italian, area=central are then applied in the same way. Every time a node is split, it is replicated and an array of all replicas is maintained. This partition splitting process expands the partitioning tree and the resulting partitions can be identified as combinations of all possible leaves in this tree. In the example from Fig. 3.2 the process results in 11 partitions and they are listed on the figure. In the cases where class or lexical nodes do not have any child node in the respective partitions it is assumed that their value is unknown – area=? in partition 2 versus area=central in partition 1 in Fig. 3.2. Before an attribute-value is applied it is checked if it already matches any of the nodes in the tree. In this way, it is made sure that no duplicates are created.

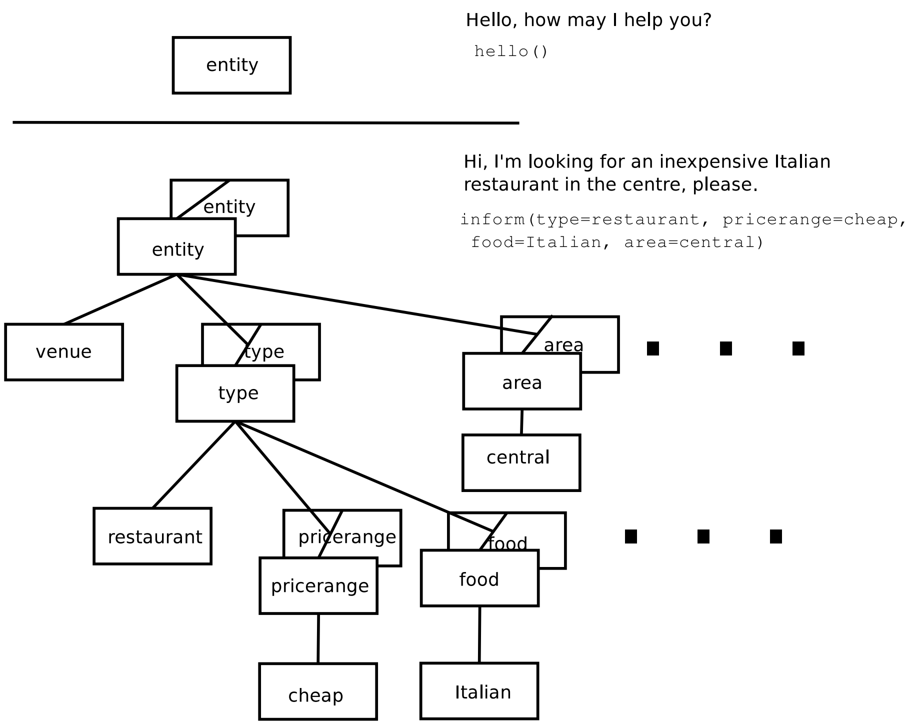
It is important to note that the number of partitions grows exponentially with the length of the N-best list input and also with the length of the dialogue itself. Therefore the length of both the N-best list and the dialogue has to be restricted to ensure tractability.

A partition can be matched against the database and that can result in a list of matching entities. The number of these entities is referred to as the partition status, which is needed for the system to make a decision in response to each system act. The list of partition statuses is given in Table 3.3.

Initial	partition has never been matched with the database
LargeGroup	partition matches more than 3 entities in the database
Group	partition matches fewer than 3 entities in the database
Unique	partition matches a unique database entity
Unknown	partition does not have matches in the database

Table 3.3: Partition status.

Similar to the user goal space, the number of possible dialogue histories is huge. In order to reduce this, the dialogue history is modelled as states associated with the partition nodes. These states are called grounding states. They model the common ground that the system and the user



Hello, how may I help you?

```
hello()
```

Hi, I'm looking for an inexpensive Italian restaurant in the centre, please.

```
inform(type=restaurant, pricerange=cheap,
 food=Italian, area=central)
```

```
Partitions:
1. entity=venue, type=restaurant, pricerange=cheap, food=Italian, area=central
2. entity=venue, type=restaurant, pricerange=cheap, food=Italian, area=?
3. entity=venue, type=restaurant, pricerange=?, food=Italian, area=central
4. entity=venue, type=restaurant, pricerange=?, food=Italian, are=?
5. entity=venue, type=restaurant, pricerange=cheap, food=?, area=central
6. entity=venue, type=restaurant, pricerange=cheap, food=?, area=?
7. entity=venue, type=restaurant, pricerange=?, food=?, area=central
8. entity=venue, type=restaurant, pricerange=?, food=?, area=?
9. entity=venue, type=restaurant, pricerange=?, food=?, area=?
10. entity=venue, type=?
11. entity=?
```

Figure 3.2: Partition splitting.

Init	Initial state
UReq	Item requested by user with expectation of an immediate answer
UInfo	Item supplied by user during formation of a query
SInfo	Item supplied by system
SQry	Item queried for confirmation by system
Deny	Item denied
Grnd	Item grounded

Table 3.4: States of the grounding model.

may have about a concept (attribute or value) at a particular stage during the dialogue. For example, the user may asked for the value of food,

Initial	default hypothesis state
Supported	hypothesis has at least one grounded node
Offered	hypothesis has been offered to the user
Accepted	hypothesis has been accepted by the user
Rejected	hypothesis has a denied node
Completed	hypothesis is complete

Table 3.5: Hypothesis status.

or inform about the value of area. The list of the grounding states and their meaning is given in Table 3.4. The transitions between these states are deterministic and are implemented via a finite state machine. For example, if the partition contains nodes cheap, hotel, area and the user says inform(type=hotel, pricerange=cheap) then the state of nodes cheap and hotel become UInfo, while node area remains in the Init state. Then, if the system says request(area) the node area transits to SQry state. These are important for the action selection process. The grounding model itself is hand-crafted, built using dialogue transition theory [105] and is expandable to most limited domain query dialogue problems.

The combination of a partition, a user act from the N-best list and the associated dialogue history forms a *hypothesis*, *i.e.,* a single member of the partitioned state space. A probability distribution over the most likely hypotheses is maintained during the dialogue and this distribution constitutes the POMDP's belief state, as explained further in Section 3.3.4. The compressed version of the grounding information included in a hypothesis is referred to as the hypothesis status, given in Table 3.5. Note that the same partition can have different grounding states associated with its nodes and this would result in two different hypothesis sharing the same partition. All possible combinations of a user act, a partition and the grounding information result in a list of hypotheses over which a distribution is maintained and updated at each turn. This is the belief state $b(h_t)$. Since it would be intractable to work with all possible hypotheses, the list is pruned to include only the most probable ones.

3.3.4 Belief update

Updating the belief state depends on four models: the observations model, the user act model, the user goal model and the dialogue history model, as outlined in Section 2.2.6. For explanation purposes it is useful to reproduce equation Eq. 2.19:

$$\begin{aligned}
&b(s^u_{t+1} = s'_u, a^u_{t+1} = a'_u, s^d_{t+1} = s'_d) \\
&\propto \underbrace{P(o_{t+1} = o' | a^u_{t+1} = a'_u)}_{\text{observation model}} \\
&\quad \underbrace{P(a^u_{t+1} = a'_u | s^u_{t+1} = s'_u, a^m_t = a_m)}_{\text{user action model}} \\
&\quad \sum_{s^u \in \mathcal{S}^u} \underbrace{P(s^u_{t+1} = s'_u | s^u_t = s_u, a^m_t = a_m)}_{\text{user goal model}} \\
&\quad \sum_{s^d \in \mathcal{S}^d} \underbrace{P(s^d_{t+1} = s'_d | s^u_{t+1} = s'_u, a^u_{t+1} = a'_u, s^d_t = s_d, a^m_t = a_m)}_{\text{dialogue history model}} \\
&\quad \sum_{a^u \in \mathcal{A}^u} b(s^u_t = s_u, a^u_t = a_u, s^d_t = s_d) \qquad (3.1)
\end{aligned}$$

where s^u_t is the user goal, a^u_t is the user action, s^d_t is the dialogue history and a^m_t is the system action at dialogue turn t.

The *user goal model* defines how the user goal evolves during the dialogue. In the HIS dialogue manager user goals are grouped into partitions. Due to the assumption that each user goal in the same partition is equally likely, the user goal model is simplified in such a way that instead of taking into account every possible goal, it only takes into account different partitions to a certain level of granularity in each particular turn. Therefore in the HIS dialogue manager, the belief is not maintained on the combination of a user goal s^u_t, a user action a^u_t and a dialogue history s^d_t as in $b(s^u_t, a^u_t, s^d_t)$ in Eq. 3.1, but rather on the combination of a partition p_t, a user action a^u_t and a dialogue history s^d_t, $b(p_t, a^u_t, s^d_t)$. In this way, the user goal model becomes

$$\sum_{s^u \in \mathcal{S}^u} P(s^u_{t+1} = s'_u | s^u_t = s_u, a^m_t = a_m) = \sum_{p \in \mathcal{P}_t} P(p_{t+1} = p' | p_t = p), \qquad (3.2)$$

where p_{t+1} is a partition from the current turn, p_t is the partition from the previous turn and $\mathcal{P}_t$ is the set of all partitions from the previous turn. Up to now, the sets of goals was fixed through the dialogue, $\mathcal{S}^u$. In the HIS system this fixed set of goals $\mathcal{S}^u$ is replaced with a dynamic set of partitions $\mathcal{P}_t$ that changes for each turn. That allows the update formula to be simpler since there is no need to sum over all possible goals, but only the ones characteristic for that turn t. Note that in Eq. 3.2 there is no need to include the dependency on system action a_t^m, since the system action already takes part in the creation of the partition set $\mathcal{P}_{t+1}$ which the partition p' belongs to.

The user goal model is further simplified by assuming that the user does not change their mind during the dialogue. In this way, there is no need to take into account that the transition may happen from any partition in the previous turn p to one in the current turn p'. On the contrary, if partition p^p represents the partition in the previous turn and the assumption is that the user goal does not change, but rather becomes more refined with new information from the user, the partition in the current turn p' is split from partition p^p. This allows for the approximation

$$\sum_{p \in \mathcal{P}_t} P(p_{t+1} = p' | p_t = p) = P(p_{t+1} = p' | p_t = p^p), \tag{3.3}$$

where $\mathcal{P}_t$ is the set of all partitions in the previous turn and $p^p \in \mathcal{P}_t$ the partition in the previous turn from which p' is split.

Eq. 3.1 therefore becomes:

$$\begin{aligned} b(&\underbrace{p_{t+1} = p', a^u_{t+1} = a'_u, s^d_{t+1} = s'_d}_{\substack{h_{t+1} = h' \\ \text{hypothesis in turn } t+1}}) \\ \propto\ &\underbrace{P(o_{t+1} = o' | a^u_{t+1} = a'_u)}_{\text{observation model}} \underbrace{P(a^u_{t+1} = a'_u | p^u_{t+1} = p', a^m_t = a_m)}_{\text{user action model}} \\ &\underbrace{P(p_{t+1} = p' | p_t = p^p)}_{\text{user goal model}} \\ &\sum_{s^d \in \mathcal{S}^d} \underbrace{P(s^d_{t+1} = s'_d | p^u_{t+1} = p'_u, a^u_{t+1} = a'_u, s^d_t = s_d, a^m_t = a_m)}_{\text{dialogue history model}} \\ &\sum_{a^u \in \mathcal{A}^u} b(\underbrace{p_t = p, a^u_t = a_u, s^d_t = s_d}_{\substack{h_t = h \\ \text{hypothesis in turn } t}}), \end{aligned} \tag{3.4}$$

where hypothesis h_t is a triple of partition p_t, user action a_t^u and dialogue history s_t^d at turn t and it takes values in $\mathcal{H}_t = \mathcal{P}_t \times \mathcal{A}^u \times \mathcal{S}^d$, with $h \in \mathcal{H}_t$ and $h' \in \mathcal{H}_{t+1}$.

The *user action model* is approximated by the dialogue act type bigram model and the matching function (k is a normalisation constant):

$$\begin{aligned} &P(a_{t+1}^u = a_u' | p_{t+1}^u = p', a_t^m = a_m) \\ &\approx k \cdot P(\mathcal{T}(a_{t+1}^u) = \mathcal{T}(a_u') | \mathcal{T}(a_t^m) = \mathcal{T}(a_m)) \cdot \mathcal{M}(a_u', p', a_m), \end{aligned} \tag{3.5}$$

where $\mathcal{T}(\cdot)$ denotes the type of the dialogue act and $\mathcal{M}(\cdot)$ indicates whether or not the user dialogue act a_u' matches the partition p' and system act a_m, $\mathcal{M}(a_u', p', a_m) : \mathcal{A}^u \times \mathcal{P}_{t+1} \times \mathcal{A}^m \to \mathbb{R}$ [106]. The bigram model determines how probable the dialogue act type is given its preceding dialogue act type in the dialogue. The matching function is a filter that gives a low probability to dialogue acts which are inconsistent with the given partition and the system act and a high probability otherwise.

The *observation model* $P(o_{t+1}|a_t^u)$ represents the probability of a particular observation given the true user act. Assuming that every act is equally probable and that every observation is equally probable (both $P(a_{t+1}^u)$ and $P(o_{t+1})$ are constant), the observation model becomes:

$$\begin{aligned} P(o_{t+1} = o' | a_{t+1}^u = a_u') &= \frac{P(a_{t+1}^u = a_u' | o_{t+1} = o') P(o_{t+1} = o')}{P(a_{t+1}^u = a_u')} \\ &\approx l \cdot P(a_{t+1}^u = a_u' | o_{t+1} = o'), \end{aligned} \tag{3.6}$$

where l is a constant.

The system gets a scored N-best list of possible user acts at turn $t+1$, $\mathcal{A}_{t+1}^u = \{\tilde{a}_u^1, \ldots, \tilde{a}_u^N\}$ with associated confidence scores $\{c1, \ldots, c_N\}$ as an input from the speech understanding component. This forms the observation $\mathbf{o}' = [(\tilde{a}_u^1, c_1), \ldots, (\tilde{a}_u^N, c_N)]^\mathsf{T}$. Due to the fact that confidence score in the HIS system is a probability distribution, the following assumption is made

$$P(a_{t+1}^u = a_u' | o_{t+1} = \mathbf{o}') = \begin{cases} c_i & \text{if } a_u' \text{ is } a_u^i \in \mathcal{A}_{t+1}^u \\ 0 & \text{if } a_u' \notin \mathcal{A}_{t+1}^u \end{cases} \tag{3.7}$$

Due to this assumption, when updating the belief in turn $t+1$ (see Eq 3.4), there is no need to sum over all possible user dialogue acts $\mathcal{A}^u$ but only over the ones that appeared in the N-best list input in the previous step $\mathcal{A}_t^u$, since the belief states $b(p_t = p, a_t^u = a_u, s_t^d = s_d)$ are zero where $a_u \notin \mathcal{A}_t^u$.

The *dialogue history model* in the HIS system is fully deterministic. The dialogue state is described as a vector $\mathbf{s}_d$ of grounding states $\rho \in \Gamma$ associated with each node η in a partition p, where Γ is the set of grounding states. The grounding model finite state machine (see Section 3.3.3) gives new grounding state ρ' from the previous grounding state ρ given user action a'_u and system action a_m. This can be denoted as $\tau(\rho, a_m, a'_u) = \rho'$, where τ is the grounding state transition function, $\tau : \Gamma \times \mathcal{A}^m \times \mathcal{A}^u \to \Gamma$. Therefore, the dialogue history model is approximated as

$$
\begin{aligned}
&P(s_{t+1}^d = \mathbf{s}'_d | p_{t+1}^u = p', a_{t+1}^u = a'_u, s_t^d = \mathbf{s}_d, a_t^m = a_m) \\
&= \begin{cases} 1 & \text{if } \tau(\mathbf{s}_d(\eta), a_m, a'_u) \text{ gives } \mathbf{s}'_d(\eta), \forall \eta \in p' \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{3.8}
$$

Due to this approximation, there is no need in Eq. 3.4 to sum over all possible dialogue histories $\mathcal{S}^d$, but rather only the ones that appeared in the previous turn $\mathcal{S}_t^d$ since the belief states $b(p_t = p, a_t^u = a_u, s_t^d = s_d)$ for dialogue histories $s_d \notin \mathcal{S}_t^d$ is zero. In fact it is only necessary to sum over dialogue histories $\mathbf{s}_d \in \mathcal{S}_t^d$ that were associated with partition p^p – the partition in the previous turn from which partition p is split. The reason for this is that the dialogue history consists of grounding states associated with each node in the partition. Since the user goal does not change over time, the dialogue histories for partition p can only evolve from the dialogue histories associated to the partition p^p. Most of the time, this is a one-to-one mapping, in which case there is only one term in the summation.

3.3.5 Policy representation

Policy representation in POMDP systems is non-trivial since each system action depends on a complex probability distribution over all hypotheses. Although grouping user goals into partitions makes it possible to maintain belief state for large scale problems, due to the size of the state space it is still prohibitive to define an action for every possible belief state – every possible distribution over hypotheses. It is also prohibitive to perform learning directly. Hence, in order to perform policy optimisation, the state space must be mapped to a smaller scale *summary space*. In the HIS system, this is achieved by heuristically mapping to summary space only a limited number of components of the belief state that are considered to be the most important for learning.

The full cycle of the action-selection process is given in Fig. 3.3. For a given belief state $\mathbf{b}$ the system needs to find an appropriate action a_m. In order to do this, it maps the belief state $\mathbf{b}$ into a summary state point $\hat{\mathbf{b}}$ for which it proposes a summary action $\hat{a}_m^i$. Then an inverse mapping is performed to obtain the action a_m in the master space.

Instead of having the complete distribution of the hypotheses, only the top two probabilities are represented in the summary space, see $\hat{\mathbf{b}}$ in Fig. 3.3. In addition, the user act type, the partition status and the hypothesis status from the top hypothesis are included in the summary space. As mentioned in Section 3.3.3, the partition status denotes how many database entities there are that match the partition, see Table 3.3. The hypothesis status is a dialogue history status variable which keeps track of dialogue progress, see Table 3.5. The user act type is simply the type of the user dialogue act that is included in the formation of the top hypothesis. In this way the (complex) belief state is represented by a point in a multidimensional space where some variables are continuous (the probability of the top two hypotheses) and some are discrete (the user act type, the partition status and the hypothesis status). The original POMDP model is thus cast to a continuous MDP model which can be solved using a grid-based approach (see Section 2.2.5).

In order to deploy the grid-based approach, a simple distance metric is defined which is Euclidean on the continuous variables and a weighted sum of δ functions on the discrete variables:

$$|\hat{\mathbf{b}}_i - \hat{\mathbf{b}}_j| = \sum_{k=1}^{2} \omega_k \cdot \sqrt{(\hat{\mathbf{b}}_i(k) - \hat{\mathbf{b}}_j(k))^2} + \sum_{k=3}^{5} \omega_k \cdot \delta(\hat{\mathbf{b}}_i(k), \hat{\mathbf{b}}_j(k)), \tag{3.9}$$

where ω_k are weights.

This distance metric allows for the summary space to be discretised into a finite number of grid points. As already mentioned in Section 2.2.5, grid points are constructed in such a way that the distance between every grid point is greater than a threshold ν and for every other point in the summary space there exists a grid point such that the distance between the two is less than ν. The weights are heuristically set to ensure that the points with different hypothesis and partition status create a new grid region ($\omega_1 = 1.0$,

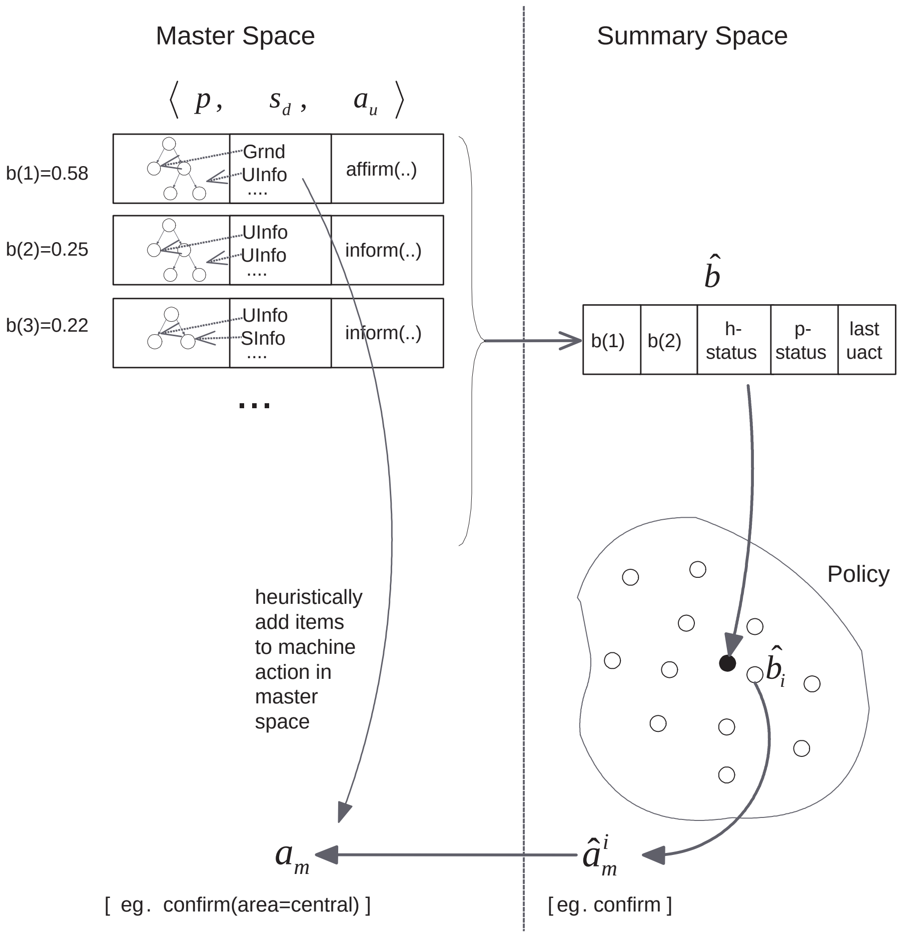


Figure 3.3: Master to summary space mapping.

$\omega_2 = 0.5$, $\omega_3 \to \infty$, $\omega_4 \to \infty$, $\omega_5 = 0.5$). This will be referred to as the *Euclidean* distance metric, thought strictly speaking it is Euclidean only on the continuous part of the space.

Discretising the summary space into a finite set of grid points allows discrete MDP algorithms to be used for optimising the policy on the grid points. Whenever a point in the summary space is visited, either during policy optimisation or policy execution, it is approximated with its nearest grid point, denoted as $\hat{\mathbf{b}}_i$ in Fig. 3.3.

Actions in the summary space refer specifically to the top two hypotheses, and unlike actions in the master space, they are limited to a small finite set, see Table 3.6. For example, if the system "offers" an entity (summary action Offer) it is reasonable to mention the entity that matches the most likely user goal, *i.e.,* the one that matches partition from the most likely hypothesis.

A system summary space action is denoted as $\hat{a}^i_m$ in Fig. 3.3. The system summary action is then mapped back into a dialogue action by adding additional information from the corresponding *master* belief state to give the required system dialogue act, denoted as a^i_m. It can be seen in Table 3.6 that summary actions require an attribute or attribute-value pairs to form the system dialogue act. The grounding information is used to determine these attribute value pairs. For example, if the summary action is Offer, then all the attribute-value pairs that the user requested are added to form the system dialogue act. The purpose of this is to assure the user that the offered entity has the requested properties. In a similar way, summary-to-master-space mapping rules are heuristically defined for each summary action.

3.3.6 Overview of HIS dialogue manager operation

At this point it is useful to summarise the HIS dialogue manager's operation. The input to the dialogue manager is a list of dialogue acts with associated confidence scores, $\tilde{a}^1_u, \tilde{a}^2_u, \ldots, \tilde{a}^N_u$, see Fig. 3.4. The attribute-value pairs from each of these dialogue acts are used in forming partitions that represent possible user goals, p^1, p^2, p^3 etc. A user act $\tilde{a}^i_u$ from the input is associated with a partition p^i and together with the corresponding grounding states $\mathbf{s}^i_d$ it forms a hypothesis h_i. Every such combination produces a list of hypotheses over which the distribution is maintained. This distribution

Summary action	Interpretation and an example master action
Greet	System greets the user hello()
Request	System requests an attribute value request(area)
Confirm	System confirms attribute values confirm(type=restaurant, pricerange=cheap)
ConfReq	System implicitly confirms attribute values and requests an attribute value confreq(type=restaurant, pricerange=cheap, area)
Select	System asks the user to select between two attribute values select(pricerange=expensive, pricerange=cheap)
Offer	System offers an entity inform(name=Char Sue, type=restaurant, pricerange=cheap)
OfferAlt	System offers an entity that has not been offered before inform(name=Peking, type=restaurant, pricerange=cheap)
QueryMore	System asks the user if there is anything else they request reqmore()
SysRepeat	System repeats the last action inform(name=Peking, type=restaurant, pricerange=cheap)
UserRepeat	System asks the user to repeat the request repeat()
Bye	System says bye bye()

Table 3.6: HIS summary actions.

represent the belief state, $b(h)$. The belief state $b(h)$ is then mapped to a point in the summary space, $\hat{\mathbf{b}}$ for which the policy provides a summary action, $\hat{a}_m$. This summary action is finally mapped to a master action, *i.e.,* the system dialogue act, by heuristically adding the grounding information from the master space.

When interacting with the user simulator, the system dialogue act is passed directly to the user simulator. In interaction with a human user, the dialogue act is passed to the natural language generator, which produces text that is then synthesised into speech.

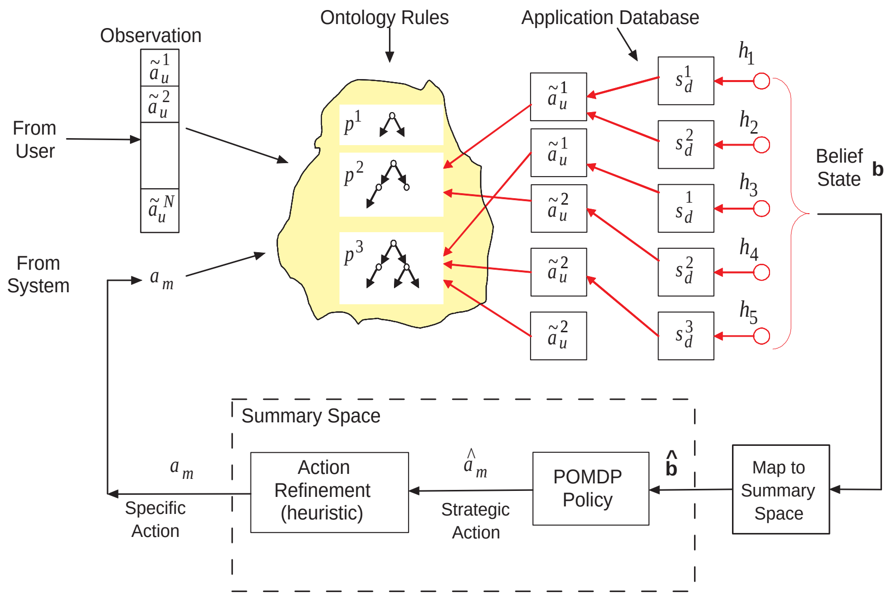


Figure 3.4: Overview of the dialogue process.

3.4 Training and evaluation with a simulated user

This section focuses on the problem of policy optimisation using a summary space. One approach to building a dialogue policy is to simply have an expert define it using rules for each part of the summary space to provide appropriate summary actions. For example, when the probability of the top hypothesis is smaller than a certain threshold, the dialogue manager should ask for a confirmation from the user in order to become more certain about the user goal. This is the Confirm summary action. In the case where the probability is very high, the system should respond to the user request and in the tourist infraction domain this normally results in suggesting an entity to the user that matches the user goal as represented by the top hypothesis. This is the Offer summary action. In a similar way, such rules can be devised for every single part of the summary space. Similar hand-crafted policies were previously deployed in other MDP and POMDP systems and provided a strong baseline [73, 90]. In order to reduce the development cost and to make the process automatic, reinforcement learning is deployed to find an optimal policy. In the HIS system, this is performed by interacting with a simulated user.

3.4.1 Policy optimisation

Policy optimisation is performed in the discrete summary space using the Monte Carlo control algorithm (Algorithm 2, see Section 2.2.2). The system maintains a set of grid points in the summary space $\{\hat{\mathbf{b}}_i\}$. At each turn in training, the nearest grid point $\hat{\mathbf{b}}_i$ to the current summary point $\hat{\mathbf{b}}$ is located using a distance measure. If the distance is greater than some threshold, $\hat{\mathbf{b}}$ is added to the set of stored grid points. The sequence of points $\hat{\mathbf{b}}_i$ traversed in each dialogue is stored in a list. Associated with each $\hat{\mathbf{b}}_i$ is a function $Q(\hat{\mathbf{b}}_i, \hat{a}_m)$ – the expected total reward obtained by choosing summary action $\hat{a}_m$ from summary state $\hat{\mathbf{b}}_i$. At the end of each dialogue, the total reward is calculated and added to an accumulator for each point in the list, discounted by γ at each step and the Q values are updated according to the accumulated rewards. The whole process is then repeated until the policy stabilises.

In the experiments that will be presented in Section 3.4.3, the exploration factor, ϵ, was fixed at 0.1. This means that the system was taking a random action 10% of the time during learning. An adequate balance between exploration and exploitation is a major problem in reinforcement learning [32]. Although the choice of ϵ is heuristic, setting the exploration factor to 10% tends to give a faster convergence for stochastic systems then setting it to a smaller value [32]. The discount factor is normally used for non-episodic tasks. However, in this case discount factor γ was fixed at 0.95 in order to favour shorter dialogues.

The reward function is defined in the following way. A positive reward of +20 will be given for successful completion of the user request deducted for the number of dialogue turns taken. Successful completion means that the system has provided the user an entity that matches the user's constraints and also provided all the information that the user requested about that particular entity. Thus, successfully completed short dialogues are favoured. Although simplistic, a similar reward function was found to be effective in other studies [10, 77, 90]. Sometimes a positive reward of +100 will be given for successful dialogue completion. This will be the case when the system needs to negotiate with the user to be able to provide an entity. For example, the user may change their mind. In such cases the dialogues may be longer.

If this is the case it will be clearly stated in the experiments, otherwise the default setting of +20 will be used.

3.4.2 User simulator and the error model

The agenda-based user simulator is used throughout this thesis for training and evaluation of the dialogue manager. In this section, its description is given based on the detailed descriptions in [90, 94, 107].

In the agenda-based user simulator, the user state is factored into an *agenda* and a *goal*. The goal ensures that the user simulator exhibits consistent, goal-directed behaviour. It has a form-like structure. At the start of each dialogue, it is randomly initialised with constraints for attribute-value pairs such as type=restaurant, food=Chinese, and the requests of attributes such as name, address, phone.

The role of the agenda is to elicit the dialogue acts that are needed for the user simulator to fulfil the goal. It stores the dialogue acts in a stack-like structure. The stack enables a temporarily storage of actions when another action of higher priority needs to be issued first. This allows the simulator to refer to previous dialogue turns at a later point.

Both the goal and the agenda are dynamically updated throughout the dialogue. These updates are controlled by *decision points*. Decision points can be deterministic or stochastic. The deterministic decision points preserve rational behaviour, while stochastic decision points enable variability in the simulated user behaviour. The stochastic decision points are controlled with parameterised probability distributions. In this way, they enable a wide spread of realistic dialogues to be generated. In addition, the simulator will relax its constraints when its initial goal cannot be satisfied. This allows the dialogue manager to learn negotiation-type dialogues where only an approximate solution to the user's goal exists. The list of parameters, their associated distribution (G for geometric and B for binomial) and their interpretation is given in Table 3.7. It is important to note that the stochastic decision points are nested in the flow-chart of the simulator control and the probabilities are conditional on this. For example, the simulator does not make a decision of removing a constraint from the goal (ConstraintRelax) with a probability of 0.667 at every dialogue turn. This decision point is only reached at particular stages of the dialogue.

The agenda-based simulated user also incorporates a patience level. In the set-up used here, the user ends the dialogue once the system repeats the same action three times in a row.

Parameter	Dist	Val.	Interpretation
InformCombination	G	0.600	Add a constraint to the goal
AddAttributeToReq	G	0.333	Add an attribute from the goal requests
YesAfterReqmore	B	0.250	Say yes without giving more info
AffirmWithAgdItem	B	0.050	When affirming provide more info
Greeting	B	0.500	Respond to greeting
ConstraintRelax	B	0.667	Remove a constraint from the goal
TellAboutChange	B	0.500	Inform about a goal change
ByeOrStartOver	B	0.333	End the dialogue or start again
DealWithPending	B	0.500	Deal with pending items on the agenda
AddEntityName	B	0.050	Provide entity name when requesting
NoAttrWithDontcare	B	0.800	Leave out attributes if their values do not matter
InformToConfirm	B	0.050	Change informs to confirms on agenda
ReqAltsAfterEntRec1	B	0.143	Request alternative
ReqAltsAfterEntRec2	B	0.143	Request alternative and change goal
RequestResponce1	B	0.200	Repeat a random constraint
RequestResponce2	B	0.200	Make up a new constraint
OverruleCorrection	B	0.100	Do not correct a misunderstanding
CorrectingAct1	B	0.450	Correct a misunderstanding with negate
CorrectingAct2	B	0.400	Correct a misunderstanding with deny
ThankAck1	B	0.100	Say thank you
ThankAck2	B	0.100	Say ok

Table 3.7: Parameter setting for the agenda-based simulated user.

The agenda-based user simulator was successfully used previously to train both an MDP and a POMDP dialogue manager [77, 90]. It was also evaluated with human judges in the following way. A corpus of simulated dialogues was generated. Since the simulated user interacts with the dialogue manager on the semantic level, both the system's and the user's actions were given an interpretation in natural language. The judges were asked to rate the naturalness of these dialogues as well as the dialogues generated with humans. The results show that they were not able to notice any difference [90]. This suggests that the agenda-based user simulator generates sufficiently complex user behaviour to be used for dialogue manager training and evaluation.

An important component of simulation is the error model. Confusions can be added to the user simulator output before it is passed to the dialogue manager so that the dialogue manager is not aware of the true dialogue act that the user simulator wanted to communicate but only receives a noisy version. This matches real situations where the input received from the speech understanding unit is corrupted with noise. The error model generates an N-best list of noisy dialogue acts. There are two error models that will be considered in this thesis.

The *uniform error model* for a fixed error rate e confuses each dialogue action in the N-best list with probability e. Otherwise, the dialogue act in the N-best list it is identical to the true one. Given that a dialogue act is confused, then there is a fixed set of rules of how each of its concepts (the type and the attribute-value pairs) are confused. The confidence score is based on how many times the same dialogue act appears in the N-best list.

The *Dirichlet error model* operates differently. It first produces the confidence scores for a fixed confusion rate e and a fixed length N of the N-best list. Then, sampling from the multinomial distribution determined by these generated confidence scores gives the position in the list where the true act is. Other dialogue acts in the N-best list are confused in the same way as for the uniform error model. More details can be found in [73].

Although the agenda-based user simulator and the error model described here are not directly trained on data, they have been sufficiently tested to confirm that they provide enough complexity to train a dialogue manager [73, 90]. The agenda-based user simulator with configuration specified above will be used through the thesis unless otherwise specified. For most experiments the uniform error model gives enough complexity and is part of the default configuration. Where necessary, the Dirichlet error model will be used and this will be made explicit.

3.4.3 Training in an incremental noise setting

When training a system to operate robustly in noisy conditions, a variety of strategies are possible. For example, the system can be trained only on noise-free interactions, it can be trained on increasing levels of noise or it can be trained on a high noise level from the outset. A related issue concerns the generation of grid points and the number of training iterations to perform. For example, allowing a very large number of points leads to poor performance as it is difficult to ensure that each point is visited enough times to acquire an adequate estimate. Conversely, having too few points leads to poor performance because different belief states are approximated with the same grid point.

The training schedule for the noise-free approach is self-explanatory. The schedule for the incremental set-up is as follows. In order to make sure that every part of the space is visited whilst minimising the number of iterations needed for training, the training starts in a noise-free environment using a small number of grid points and continues until the performance of the policy levels off. The resulting policy is then taken as an initial policy for the next stage where the noise level is increased, the number of grid points is increased and the number of iterations is increased. This process is repeated until the highest noise level is reached. This approach is motivated by the observation that a key factor in effective reinforcement learning is the balance between exploration and exploitation. In POMDP policy optimisation, which uses dynamically allocated grid points, maintaining this balance is crucial. Also, obtaining a good policy for a noise-free environment is typically faster than for a noisy environment. Therefore, the approach is to essentially adapt the policy obtained in a noise-free environment to different noise levels rather than just training it on a random noise level.

The noise introduced by the simulator provides an implicit mechanism for exploring the space. Each time the noise is increased, different areas of the state space will be visited and hence the number of available grid points must also be increased. At the same time, the number of iterations must be increased to ensure that all points are visited a sufficient number of times. In practice, around 750 to 1000 grid points proved to be sufficient and the total number of simulated dialogues needed for training was around $100,000$.

A second issue when training concerns the length of the N-best input. A limiting factor here is that the computation required for N-best training is significantly increased since the rate of partition generation in the HIS model increases exponentially with N. Therefore only 2-best lists were considered when training and testing in noisy conditions. Fig. 3.5 and 3.6 show the average dialogue success rates and rewards for 3 different policies: a hand-crafted policy (hdc), a policy trained on noise-free conditions (noise free) and a policy trained using the incremental scheme described above (increm). Each policy was tested using 2-best output from the simulator across a range of error rates.

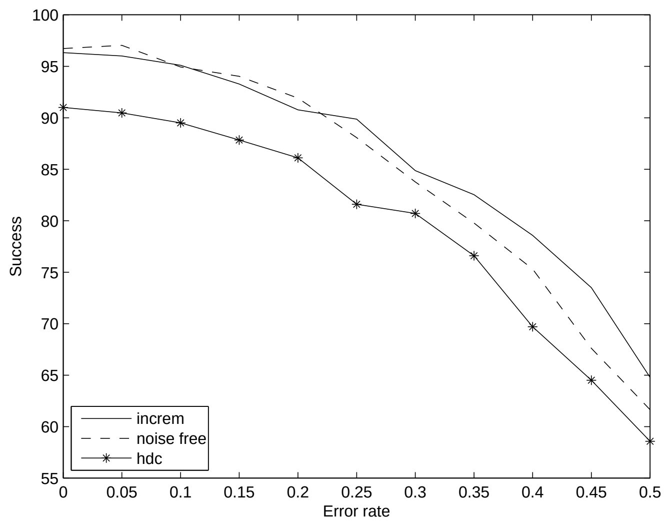


Figure 3.5: Average simulated dialogue success rate as a function of error rate for a hand-crafted (hdc), noise-free and incrementally trained (increm) policy.

As can be seen in Fig. 3.5 and 3.6, both the trained policies improve significantly on the hand-crafted policies. Furthermore, although the average rewards are all broadly similar, the success rate of the incrementally trained policy is significantly better at higher error rates.

3.5 Evaluation via a user trial in noisy conditions

It is important to check whether these benefits obtained in simulations also translate to improvements with human users. As a result, the HIS dialogue manager with a trained POMDP policy was evaluated in a user trial to

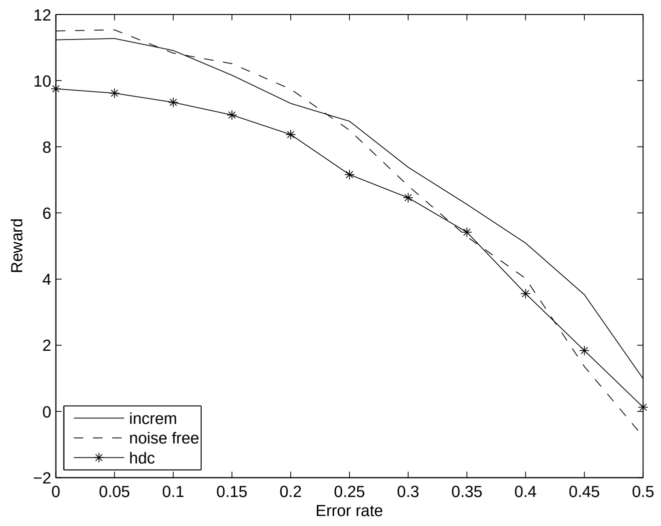


Figure 3.6: Average simulated dialogue reward as a function of error rate for a hand-crafted (hdc), noise-free and incrementally trained (increm) policy.

compare it to an MDP dialogue manager on the same task in noisy conditions. Noise was artificially generated and added to the user's speech in order to induce speech understanding errors and then measure how different approaches to statistical dialogue management deal with erroneous input. The set-up consisted of an ATK-based speech recogniser, a Phoenix-based semantic parser [108], a diphone based speech synthesiser [109] and the dialogue manager described in Section 3.3.

The HIS-POMDP policy (HIS-TRA) that was incrementally trained on the simulated user using 2-best lists was tested in a user trial together with a hand-crafted HIS-POMDP policy (HIS-HDC). In addition, an MDP-based dialogue manager [90] was tested. Since considerable effort had been put in optimising this system, it served as a strong baseline for comparison. Again, both a trained policy (MDP-TRA) and a hand-crafted policy (MDP-HDC) were tested.

3.5.1 Trial set-up

For the user trial, 36 subjects were recruited (all British native speakers) that never before had interacted with any of the systems that were being evaluated. Each subject was asked to imagine themselves to be a tourist in a fictitious town called Jasonville and to try to find particular hotels, bars, or restaurants in that town. Each subject was asked to complete a set of predefined tasks where each task involved finding the name of a venue satisfying a set of constraints such as food type is Chinese, price-range is cheap, etc., and getting the value of one or more additional attributes of that venue such as the address or the phone number.

The noise was artificially generated and mixed with the microphone signal. In addition it was fed into the subject's headphones so that they were aware of the noisy conditions. The noise levels correspond to signal to noise ratios (SNRs) of 35.3 dB (low noise), 10.2 dB (medium noise), or 3.3 dB (high noise).

3.5.2 Results

In Table 3.8, general statistics of the corpus resulting from the trial are given. The semantic error rate is based on substitutions, insertions and deletions errors on semantic items. When tested after the trial on the transcribed user utterances, the semantic error rate was 4.1% whereas the semantic error rate on the ASR input was 25.2%. This means that 84% of the semantic error rate was due to errors in the ASR.

Word error rate	32.9
Semantic error rate	25.2
Semantic error rate transcriptions	4.1

Table 3.8: General corpus statistics.

Tables 3.9 and 3.10 present average success rates and average reward, comparing the two HIS dialogue managers with the two MDP baseline systems. For the success rates, the standard deviation (std. dev.) is also given, assuming a binomial distribution. The success rate is the percentage of successfully completed dialogues. A task is considered to be fully completed when the user is able to find the venue he is looking for and gets all the additional information he asked for; if the task has no solution and the system indicates to the user no venue could be found, this also counts as full completion. A task is considered to be partially completed whenever the correct venue was been given, regardless of whether the further information was provided. The results on partial completion are given in Table 3.9, and the results on full completion in Table 3.10. To mirror the reward function used in training, the performance for each dialogue is computed by assigning a reward of 20 points for full completion and subtracting 1 point for the number of turns up until a successful recommendation (*i.e.,* partial completion).

Partial task completion statistics			
System	Success (std. dev.)	#turns	Reward
MDP-HDC	68.52 (4.83)	4.80	8.91
MDP-TRA	70.37 (4.75)	4.75	9.32
HIS-HDC	74.07 (4.55)	7.04	7.78
HIS-TRA	84.26 (3.78)	4.63	12.22

Table 3.9: Success rates and performance results on partial completion.

Full task completion statistics			
System	Success (std. dev.)	#turns	Reward
MDP-HDC	64.81 (4.96)	5.86	7.10
MDP-TRA	65.74 (4.93)	6.18	6.97
HIS-HDC	63.89 (4.99)	8.57	4.20
HIS-TRA	78.70 (4.25)	6.36	9.38

Table 3.10: Success rates and performance results on full completion.

The results show that the trained HIS dialogue manager significantly outperforms both MDP based dialogue managers. For success rate on partial completion, both HIS systems perform better than the MDP systems.

It is difficult to directly compare these results with the evaluation on the user simulator (Figs. 3.5 and 3.6), since the error rate that the uniform error model produces does not directly relate to the semantic error rate (Section 3.4.2). However, calculating the semantic error rate on the simulated data showed that the semantic error rate of 0.25 relates to the error rate of 0.40 in the uniform error model. When comparing the results form Table 3.4.2 with the evaluation on the simulated user (Figs. 3.5 and 3.6) for 0.40 error rate, it can be seen that the average success of the trained HIS policy and the average reward of the hand-crafted HIS policy are similar to the real-user evaluations. However, the average reward of the trained policy is much higher and the average success of the hand-crafted policy is lower than when tested with the simulated user. This suggests that the simulated user is able to accurately predict the rankings of the systems but not the absolute performance.

3.5.3 Subjective results

In the user trial, the subjects were also asked for a subjective judgement of the systems. After completing each task, the subjects were asked whether they had found the information they were looking for (yes/no). They were also asked to give a score on a scale from 1 to 5 (best) on how natural/intuitive they thought the dialogue was. Table 3.11 shows the results for the 4 systems used. The performance of the HIS systems is similar to the MDP systems, with a slightly higher success rate for the trained one and a slightly lower score for the hand-crafted one.

System	Succ. Rate (std. dev.)	Score
MDP-HDC	78 (4.30)	3.52
MDP-TRA	78 (4.30)	3.42
HIS-HDC	71 (4.72)	3.05
HIS-TRA	83 (3.90)	3.41

Table 3.11: Subjective performance results from the user trial.

The discrepancy between the objective and subjective user results (Tables 3.10 and 3.11) demonstrate the limitations of controlled tests. Users' average perception that the systems were more successful than the objective measure may be ascribed to the users not checking whether the entity that the system offer exactly matches the task specification. It seems that they perceived HIS-TRA to be more successful than the MDP systems, which is in line with with the objective results. However, more dialogues are needed to establish statistical significance. Their average scores on naturalness are not very informative, since the frame of reference varies sharply across different subjects. This altogether makes it very difficult to use these results to establish a valid comparison, and it points out the need to evaluate the system with real users in a real situation performing a much larger number of dialogues.

3.6 Summary

This chapter has given an overview of the HIS system and a detailed description of the HIS dialogue manager. The process of policy optimisation with a simulated user in an incremental noise setting was explained and evaluation results from a trial with human users were given.

The results from the user trial show the superiority of the POMDP approach compared to the MDP approach for dialogue modelling, but they also point to some deficiencies of the HIS system. Although the original HIS system is able to deal with a large limited-domain dialogue task, in order to achieve tractability in training, the length of the N-best list is restricted to 2. This is mainly due to the exponential growth of the number of partitions in high error rates with large N-best lists and the lack of an effective partition pruning mechanism. This is a considerable limitation since large N-best lists provide more information about the possible user request. In addition, the exponential partition growth also places a constraint on the supported dialogue length, which is not desirable in real-world systems. Moreover, the original state representation does not support more complex dialogues, such as dialogues with user requests containing logical combinations of constraints. Also, although the system was trained to deal with negotiation-type dialogues, *e.g.,* a dialogue in which the user asks for an alternative entity from the offered one or the user changes their mind, the performance on these dialogues is typically much worse then when the user goal stays constant. This is due to the fact that the belief update formula Eq. 2.19 in Section 3.3.4 is based on the assumption that the user goal does not change.

Another issue is policy optimisation. There are clear problems in optimising the policy using a summary space, which is drastically downscaled, as well as in optimising it via interaction with a simulated user. Downscaling the space inevitably leads to problems in mapping back to the master space. Also, enlarging the state space increases the number of iterations needed for training. This is undesirable because the policy should ideally be trained on real users rather than a simulated user. Therefore, a faster policy optimisation method is needed.

These issues will be addressed in the following chapters. Chapter 4 explains how the partition representation can be improved to deal effectively with the exponential growth of partitions in order to support large N-best lists inputs from the speech understanding unit, dialogue domains with complex ontologies and dialogues with a large number of dialogue turns. The problem of policy optimisation will be addressed in two ways. Firstly, in Chapter 5 the problem of the inverse mapping will be examined via a back-off mechanism that guarantees the optimal back-off action selection. Secondly, a policy optimisation method where the Q-function is modelled as a Gaussian process will be investigated. Chapter 6 introduces this method and examines its potential to speed up the policy optimisation process. Chapter 7 then further investigates how this model for the Q-function can be used for adaptation to different user profiles.

Chapter 4

Extended state representation

4.1 Introduction

In order to exploit the HIS system's capability to model dependencies between different attribute-value pairs (which is in contrast with the BUDS system, for example), the standard model is extended to include an explicit representation of complements. Similar to [72, 110], the partitions are formed using not only the attribute value pairs from the user input, but also their complements. In this way, the coverage of potential user goals is improved. This is particularly useful for more complex dialogue structures, where the user goal evolves and changes during the dialogue, influenced by the system's responses. Moreover, the notion of complements allows a variant of first order logic to be incorporated, *i.e.,* the user can use negations, conjunctions and disjunctions to communicate with the system and the system can use quantifiers to express the result of the user query.

Since the uncertainty in the user input is dealt with by taking into account all partitions that can be referred to by the user input, the number of possible partitions grows exponentially as the dialogue progresses. This results in computational difficulties, especially in domains where relatively long dialogues are expected. It also limits the length of the N-best list of hypothesised user dialogue acts input to the dialogue manager, which is crucial for robust belief monitoring in noisy conditions [102]. This chapter shows how the explicit notion of complements allows an efficient pruning technique to be implemented which enables arbitrarily long N-best lists of input acts and arbitrarily long dialogues to be supported, while preserving the most probable user goals. This new representation resolves many of the computational problems found in the shared node representation used in the original HIS system (outlined in Section 3.3.3) and at the same time extends its expressive power.

4.1.1 Application domain

The extensions to the state representation that are presented in this chapter have made it possible to apply the HIS formalism to a much larger real-world dialogue task: tourist information for Cambridge. In this system, which is called CamInfo, the user can ask for a restaurant, hotel, bar, museum or another tourist attraction. The database consists of approximately 500 entities, each of which has up to 10 attributes that the user can query. The full description of the CamInfo domain ontology is given in Appendix C.1.

4.1.2 Dialogue tasks requiring negotiation

Including complements in the dialogue representation allows the dialogue manager to provide better support for dialogue tasks requiring negotiations. As noted in Section 3.4.2, negotiation-type dialogues are dialogues where the user does not have a firm goal, but their goal is influenced by the system's response and can potentially change over the course of a dialogue. For example, every time the system offers information, the user can change their mind and ask for something else. Another example is when the system informs the user that there is no matching entity in the database for the user's request. In this case, the user can relax the constraints and try to find something else. Normally, in task-based dialogue systems such scenarios are not supported. The Hidden Information State is also based on the idea that the user goal does not change. Moreover, this constraint is not only incorporated in the belief update as mentioned in Section 3.3.4, but more importantly in the state representation.

For example, consider the attribute value pairs name=Char Sue, type=restaurant and pricerange=cheap. In the representation without complements this would result in four partitions: one that contains name=Char Sue, type=restaurant and pricerange=cheap (partition 1), another that contains type=restaurant and pricerange=cheap (partition 2), one with just type=restaurant (partition 3) and one with just pricerange=cheap (partition 4). Thus, if the system makes an offer inform(name=Char Sue,type=restaurant,pricerange=cheap), this matches partition 1 most closely. However, if the user asks for an alternative, the user goal could be represented by any of the partitions 2, 3, and 4 despite the fact that these could all yield the same entity Char Sue from the database. Preventing this to ensure that a genuine alternative is offered to the user is difficult and requires ad hoc hand-crafting. Hence, although sharing nodes among partitions allows for a vast number of partitions to be efficiently represented, it is difficult to identify and remove partitions that are represented in such a way. Including information about complements has the potential to alleviate this problem.

4.2 Explicit representation of complements

As explained in Section 3.3.3, a partition in the standard HIS model represents a realisation of the ontology tree with a unique value in each atomic, lexical or class node. In the extended representation of dialogue state, a partition retains unique values for class and lexical nodes, for example type, area or food. However, atomic nodes are represented as a set of boolean indicators for each possible value from the ontology. For example, "expensive=$\bot$ moderate=$\top$ cheap=$\top$", represents one atomic node where $\top$ and $\bot$ are logical true and false and the example matches user goals with moderate and cheap price range but not expensive. This allows a wider coverage of possible user goals. However, a potential drawback of this set representation is the need to enumerate every value that an attribute can take. This can be a problem for domains that have classes with high cardinality. For example, if one were building an automated telephone dialler, an attribute such as name could take tens of thousands of values. An alternative and more compact approach is to represent atomic nodes as a disjunction of the values which are true or a conjunction of the negation of the values which are false. In this way, the same expressibility is retained without the need to explicitly enumerate every possible value that an atomic node can take. For this purpose, standard logic notation is used where $\vee$ represents "or", $\wedge$ represents "and" and $\neg$ represents "not". An example of different representations of the same atomic node is given in Table 4.1.

Representation	Atomic node for Lexical node food						
Set	Chinese $\top$	English $\perp$	Indian $\top$	Italian $\perp$	Japanese $\perp$	French $\perp$	Thai $\perp$
Disjunctions	Chinese $\vee$ Indian						
Conjunctions	$\neg$English $\wedge$ $\neg$Italian $\wedge$ $\neg$Japanese $\wedge$ $\neg$French $\wedge$ $\neg$Thai						

Table 4.1: Different representations of the same atomic node in a partition.

The extended representation of atomic nodes allows for an extension in both the syntax and semantics of attribute-value pairs. The notation attribute!=value is introduced, meaning attribute can be anything but value. For example, pricerange!=expensive matches any atomic node where expensive=$\perp$ in the set representation. In the representation with conjunctions, it matches any atomic node that contains $\neg$expensive. In the representation with disjunctions, it matches any atomic node that does not contain expensive.

4.3 Partitioning process

In contrast to the original partitioning formulation where partitions have shared nodes, in the extended representation the partitions are autonomous with a *parent-child* relationship which keeps track of the order they are created. Formally, partitioning is the process of applying an attribute-value pair $\alpha = \beta$ to a partition p that contains node α and creating its child partition c. In the ontology, α is either a class or a lexical node and β is an atomic node. In the partition p, node α has a child atomic node η that has all possible values that attribute α can take. During the partitioning process, the value β in the atomic node η is set to false and the partition c is a copy of p in which β of the corresponding node is set to true.

In order to apply the attribute-value pair $\alpha = \beta$ to an existing set of partitions, one must first ensure that there is a partition that contains node α. For attribute α, the list of *superiors* is defined as all attribute-value pairs $\alpha_i = \beta_i$ where α_i are class nodes on the path from the node α to the root of the ontology tree, and β_i are the values of their child atomic nodes that enable the attribute expansion leading to the occurrence of α in the tree. For example, for attribute-value pair food=Italian the list of superiors is type=restaurant, entity=venue (see Table 3.2 in Section 3.3.2). The ontology automatically generates this list for each attribute α, so that they can be applied prior to applying $\alpha = \beta$. In this way, one can ensure that there is a partition with node α before $\alpha = \beta$ is applied.[1]

The partitioning process starts by applying the list of attribute-value pairs from the N-best user input to the initial partition, which is just the root of the ontology tree. The process is then recursively repeated. After the process, an ordered tree of partitions is created, where the order indicates when each partition was created. Attribute-value pairs from the system act are also applied during this partitioning process to ensure that the belief state includes attributes which have been introduced by the system as well as by the user.

It is important to note that this process guarantees that all created partitions are unique. This is achieved by checking whether a partition contains node α with child node η where β set to false before $\alpha = \beta$ is applied to that partition. If it does contain β set to false, then $\alpha = \beta$ must have already been used and should not be applied again to that partition.

A step-by-step example of the partitioning process is given in Figs. 4.1-4.2. The final tree of partitions represents the partitions that are created from the following attribute-value pairs: entity=venue, type=restaurant, area=central, food=Italian and pricerange=cheap. The ontology from Table 3.2 is used to determine the valid combinations. Therefore, there is no combination that involves type!=restaurant and food=Italian, since the lexical node food is specific to class node type in which the atomic child value restaurant is set to true.

4.4 Logical expressions for negotiation-type dialogues

The explicit representation of complements in partitions improves the model in a number of ways.

Firstly, it makes the error recovery process much simpler. For example, if attribute-value pair $\alpha = \beta$ occurred in the N-best user input due to a recognition error, and it turns out later in the dialogue that the user does not want β, then the user modelling component will automatically increase the probability of the partition that contains $\neg\beta$. In this way, even if the

[1]This mechanism is essential for dialogues where the user takes the initiative. For example, System: "How may I help you?", User: "I want some Italian food".

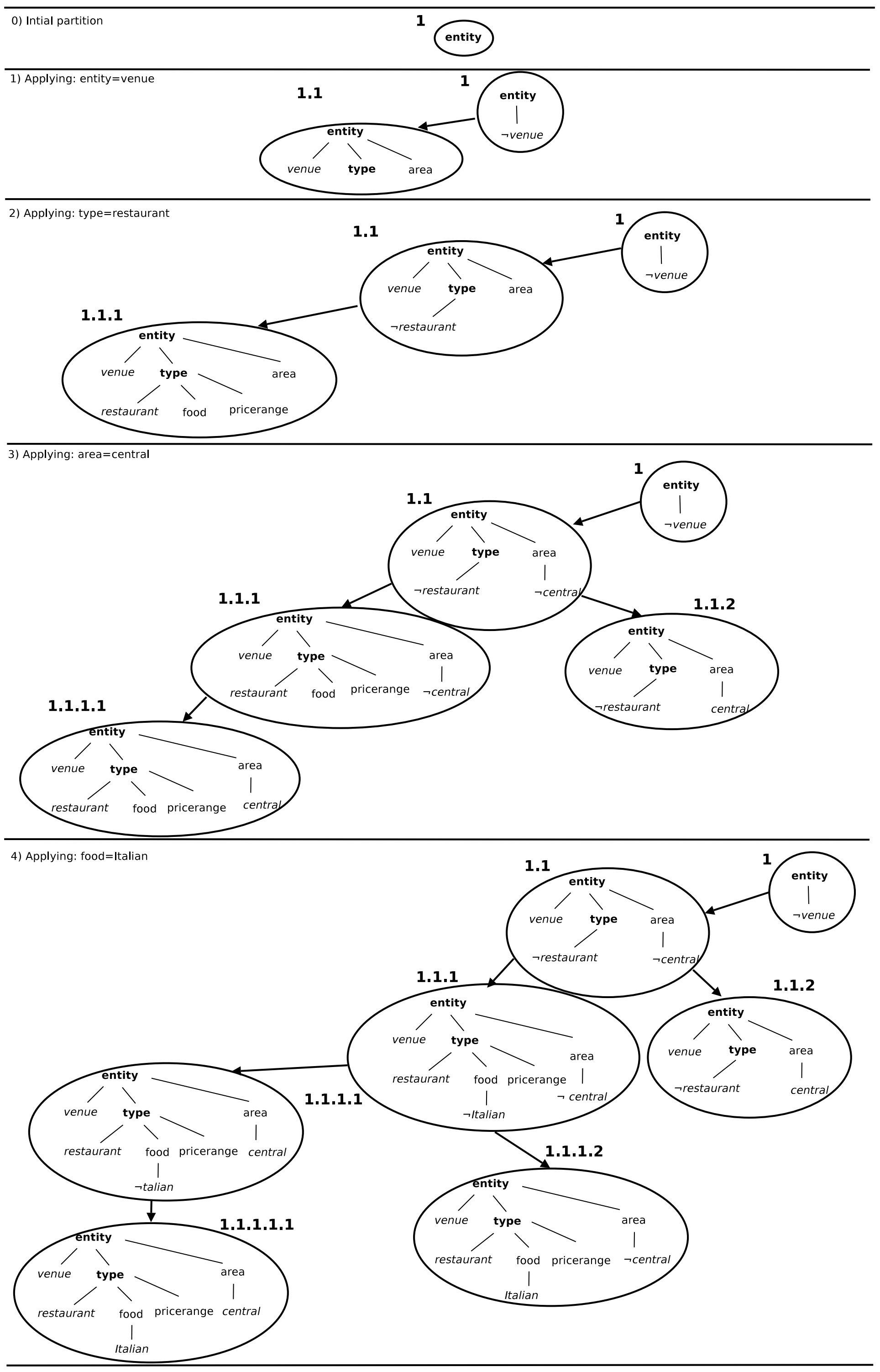


Figure 4.1: Step-by-step partitioning process.

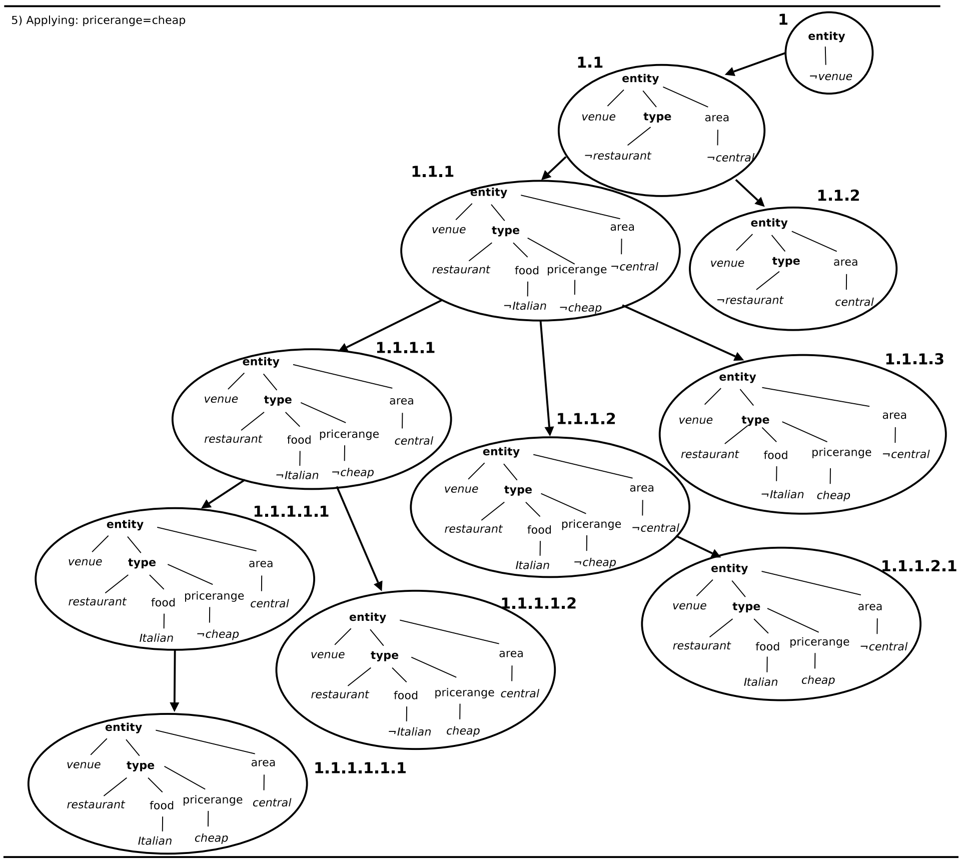


Figure 4.2: Step-by-step partitioning process (cont.).

system does not know exactly what the user wants for attribute α, the knowledge that the user does not want β is explicitly represented and the true user goal will be in the partition that has $\neg\beta$.

Secondly, this representation is particularly useful when the user goal evolves and changes during the dialogue. For example, if the user wants a Chinese restaurant in the centre, the system may offer *"Charlie Chan is a Chinese restaurant in the centre"*, which is represented at the dialogue act level by inform(name=Charlie_Chan, type=restaurant, food=Chinese, area=central). When the system makes such an offer, the partitioning results in some partitions containing name=Charlie_Chan and others containing name!=Charlie_Chan. In a real dialogue, a user might want to have more options and may ask *"Do you have anything else?'*, corresponding to the dialogue act reqalts(). Based on this, the user action model will increase the probability of partitions which have name!=Charlie_Chan and decrease the probability of partitions that have name=Charlie_Chan.

4.4.1 Quantifiers in the system's response

Using the notion of complements, the system can provide more accurate responses to the user. Referring back to the example from the previous section, if the user wanted something other than Charlie Chan's, it may turn out that the partition with name!=Charlie_Chan, type=restaurant, food=Chinese and area=central does not have any matching entries in the database. In that case, the system may respond with inform(name=none, type=restaurant, food=Chinese, area=central, name!=Charlie_Chan), meaning *"There is no restaurant that serves Chinese food and is in the centre and isn't Charlie Chan"*, or in a more natural form *"Charlie Chan is the* ***only*** *Chinese restaurant in the centre"*. An example of a Cambridge tourist information dialogue that utilises such expressions is given in Table 4.2.

During the course of a dialogue with a high rate of speech recognition errors, the system might not know what the user wants, but it might be confident about what the user does not want. Coming back to the previous example, the speech recogniser might output *"No, I don't want the central area, I want..."* corresponding to deny(area=central) or inform(area!=central), but the actual value for the attribute area might not be recognised. Then, after applying the user action model, the partition containing name!=Charlie_Chan, type=restaurant, food=Chinese and area!=central might end up having the

System	hello()
User	inform(=hospital)
System	inform(name="Addenbrooke's Hospital", type=amenity, amtype=hospital)
User	request(area)
System	inform(name="Addenbrooke's Hospital", area=addenbrookes)
User	inform(area!=addenbrookes)
System	inform(name=none, type=amenity, amtype=hospital, name!="Addenbrooke's Hospital")
User	request(addr)
System	inform(name="Addenbrooke's Hospital", addr="Cambridge University Hospitals NHS Foundation Trust, Hills Road")
User	bye()

Table 4.2: Dialogue with negations in system's response.

highest probability. If such a partition does not have any matching entries in the database, the system may inform the user about it rather than trying to find out the exact value of attribute area that the user wants. The resulting system dialogue act would then be inform(name=none, type=restaurant, food=Chinese, area!=central), meaning *"There is no restaurant that serves Chinese food and is not in the central area"*, or alternatively *"**All** Chinese restaurants are in the centre"*.

These two examples demonstrate the capability that the explicit representation of complements provides for expressing existential and universal quantifiers in the system's response. This can make the dialogue more efficient as there is no need for the system to try to resolve every single constraint that the user might have, since knowing just what the user does not want might be sufficient to provide an acceptable response.

4.4.2 Disjunctions and conjunctions in the user's request

It was mentioned previously that atomic nodes in partitions can be represented in terms of conjunctions or disjunctions of values in partitions. So far, only the case when the user has one specific value for each attribute in mind was considered, and any cases of multiple values in partitions come from confusions caused by recognition errors. However, this representation also allows a richer expression of the user's constraints. Consider the example from Fig. 4.1-4.2 where instead of Italian the user asks for an Italian or Indian restaurant, inform(type=restaurant, area=central, food=Italian$\vee$Indian,

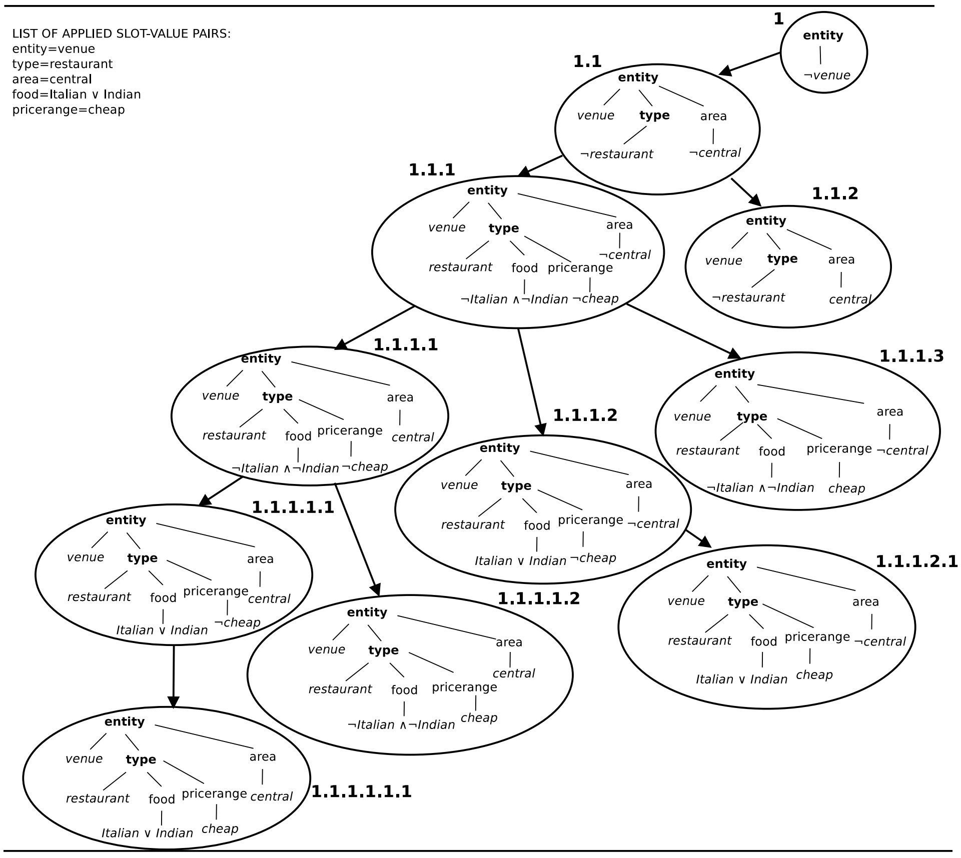


Figure 4.3: Representing disjunctions in a tree of partitions.

pricerange=cheap). Partitions can directly represent this, see Fig. 4.3 where partition 1.1.1.1.1.1. matches the user request. Note that in this case it is more effective to use the representation in terms of disjunctions and conjunctions instead of the set representation.

An example of a Cambridge tourist information dialogue where the user makes use of disjunctions is given in Table 4.3. Since every entity has only one value for each attribute, for example a hotel can only have a certain number of stars, conjunctions such as food=Italian∧Indian are not be applicable. However, conjunctions in terms of negations are possible and effectively represent disjunctions of negated values; food!=Italian ∧ food!=Indian or alternatively food!=Italian∨Indian. In an example similar to the one before, if the user asked for any restaurant but Italian or Indian, inform(type=restaurant, area=central, food!=Italian∨Indian, pricerange=cheap), the resulting partition tree would be the same as in Fig. 4.3 except that in this case the true

System
User
	hello()
	inform(=restaurant, **food=Thai $\vee$ Indian**, pricerange=cheap))

System
 	
User
	inform(name="The Rice Boat", type=placetoeat,
	eattype=restaurant, pricerange=cheap, food=Indian)
	reqalts(food=Thai)

System
User
	inform(name=none, type=placetoeat, pricerange=cheap, food=Thai)
	request(name="The Rice Boat", addr)

System
User
	inform(name="The Rice Boat", addr="37 Newnham Road")
	bye()

Table 4.3: Partitions with disjunctions and conjunctions.

user act matches partition 1.1.1.1.2. There an example of a Cambridge tourist information dialogue that makes use of conjunctions in Table 4.4.

System
User
	hello()
	inform(=hotel, **stars!="4",stars!="5"**)

System
 	
User
	inform(name="Express by Holiday Inn Cambridge",
	type=placetostay, staytype=hotel,stars="2")
	reqalts()

System
 	
User
	inform(name="Royal Cambridge Hotel", type=placetostay,
	staytype=hotel,stars="3")
	request(area)

System
User
	inform(name="Royal Cambridge Hotel", area=riverside)
	bye()

Table 4.4: Dialogue with conjunctions in the user's request.

4.5 Expressive power of the extended representation

Including the complements in the representation allows attribute-value pairs that contain negations such as $\alpha = \neg\beta$ to be a part of the user goal. In addition, it allows disjunctions of values for a particular attribute: $\alpha = \beta_1 \vee \ldots \vee \beta_k$ to be represented. Conjunctions, however, can only be represented as $\alpha = \neg\beta_1 \wedge \ldots \wedge \alpha = \neg\beta_n$, which is equivalent to negation of disjunctions $\alpha = \neg(\beta_1 \vee \ldots \vee \beta_n)$. In order to represent arbitrary conjunctions of values for an attribute, $\alpha = \beta_1 \wedge \ldots \wedge \beta_2$, a further extension is needed. Consider the set representation of the atomic node in Table 4.1. If the set of values were extended to the power set of values, then all possible conjunctions could be represented. This would drastically increase the memory requirements needed for the storing of partitions. Instead of the set representation for the atomic nodes, a better choice would therefore be to use the representation in terms of conjunctions and disjunctions (as in Fig. 4.3).

The partition representation proposed here implicitly supports conjunctions between different attribute-value pairs, $(\alpha_1 = \beta_1) \wedge \ldots \wedge (\alpha_k = \beta_k)$. However, disjunctions of different attribute-values, $(\alpha_1 = \beta_1) \vee \ldots \vee (\alpha_k = \beta_k)$, cannot be represented in the same partition. The only way to represent this in the proposed partition representation is to have a separate partition for each conjunct. However, that is not desirable since each partition represents one goal rather than parts of it. An alternative way to support the representation of disjunctions of different attribute-values in a user goal would be to maintain the distribution on the power set of partitions instead of the set of partitions. While this would allow the user request to be model any form of first order logic, it dramatically increases the computational complexity.

Finally, it is important to note the extent to which the representation of the user goal adopts a closed-world assumption. Under the closed-world assumption, if something is not known to be true it is assumed to be false [111]. In the approach adopted here, the probability distribution is maintained over all user goals throughout the dialogue. Therefore, there is a probability that one user goal is true and at the same time there is a probability that its complementary user goal is true. Given the definition of the ontology, any user goal is possible with some probability at any point in the dialogue, regardless of the structure of the tree of partitions. At different turns it may belong to different partitions, but it is always present in the representation. The user goals that cannot be expressed under the assumed ontology are ignored.

4.6 Relation to dynamic semantics

Dynamic semantics is a broad field of logic in natural language semantics that deals with the problem of representing information that grows over time. The emphasis is not on whether the meaning of an utterance is true or false, but instead on how it changes the context [112].

Discourse is a form of natural language that extends over several utterances [113]. Dialogue can be seen as an example of discourse. Dynamic semantics allows the representation of discourse to be built incrementally [112].

This is very similar to the main idea of partitioning where the representation of the user goal is built incrementally, using the user's and the system's utterance. The notion of context in dynamic semantics terminology corresponds to the representation of the user goal in the HIS framework.

In dynamic semantics, the information is partial and may not be true [112]. This is established through the use of operators for *true*, *false* and *maybe* [114]. This is crucially different to the belief state representation adopted here. In the belief state representation, there is a distribution over all possible user goals, but each user goal on its own only has a representation of the concepts that are true or false (see Table 4.1). The assumption is that the user knows what they want at every dialogue turn. Therefore, the user utterance *I might go for a Chinese restaurant* is represented in the same way as *I'd like to go for a Chinese restaurant.* The system, however, does not know the true user goal and therefore maintains a distribution of over all possible user goals through out the dialogue.

Another important aspect of dynamic semantics is anaphora resolution [115]. Anaphora is a reference to something that has been mentioned before [113]. Anaphora resolution is a way of determining what a particular anaphora refers to. Various anaphora, *e.g.,* pronoun, tense and presupposition, can be resolved using discourse representation theory [116]. Discourse referent structures are created for different utterances and the referents between these structures are matched to resolve anaphora [115]. In a simplified form compared to discourse representation theory, the representation of partitions can support anaphora resolution when the dialogue act is matched with the partition. For example, consider the case where the user requested a restaurant in the centre inform(type=restaurant, area=centre) and system offered inform(name="Char Sue",type=restaurant,area=centre). Later in the dialogue the user can refer to that restaurant by mentioning its area for instance request(phone,type=restaurant,area=centre). That dialogue act matches the partitions that contain area=central. Given that name="Char Sue" was already the topic of conversation, the partition that has both name="Char Sue" and area=central will have the highest probability. Note that this is one of the main differences with alternative approaches like the Bayesian Update of Dialogue State (discussed in Section 2.5), where different concepts that are not directly related to each other in the domain ontology, are assumed to be independent.

The final aspect of dynamic semantics that is considered here is discourse coherence. Dynamic semantics aims to rank different discourse interpretations and to measure their degree of coherence [115]. It is important to note that the belief state implicitly provides such a measure. At every dialogue step all possible hypotheses are created and they are ranked by their belief. If there is a mismatch between the dialogue act and the partition from the same hypothesis, it is given a low probability by the dialogue act matching model (Eq. 3.5 in Section 3.3.4). This mismatch can be interpreted as a lack of coherence, which results in the hypothesis having a lower rank.

4.7 Pruning process

One of the biggest problems with applying the Hidden Information state model to real-world domains is that the number of partitions grows exponentially as the dialogue progresses. This results in a serous computational limit which must be addressed for the model to be widely applicable. The approach proposed here is to prune the hypotheses so that a maximum number of partitions are maintained at any time.

In order to illustrate this problem, a simple experiment was performed on the Cambridge tourist information domain (Appendix C.1). 3000 dialogues were generated in interaction with the agenda-based simulated user as described in Section 3.4.2. The Dirichlet error model (Section 3.4.2) was used to generate 10-best list at 40% error rate. The reason the Dirichlet error model was used here, in contrast to the uniform error model in the default setting in Section 3.4.2, is that the uniform error model tends to give a very high confidence to the true dialogue act if the N-best list is large, making it easier for the dialogue manager to infer the true user goal.

Each dialogue was restricted to have a maximum of 100 turns. In Fig. 4.4 the average number of operating partitions is shown for first 22 dialogue turns. In addition, the average time in seconds that is taken for the belief update in each turn is given at each point in the graph[1]. In order for the system to run in real time, the total response time (including decoding and generation) should not be larger than 1 second, which means that the belief update time should be kept well below that threshold. It is clear that some

[1]The runtime results are obtained on an 8-core Intel Xeon 2.83GHz processor and 24Gbytes RAM.

kind of pruning technique is needed.

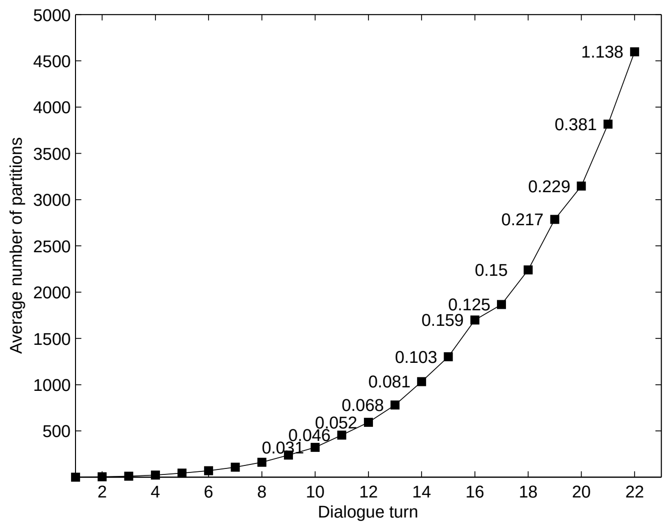


Figure 4.4: Average number of partitions for dialogue turn. The values on the curve denote the average time to perform the belief update in seconds.

4.7.1 Partition recombination

Pruning should be performed in such a way that the distance between the belief state before and after pruning is minimal [117]. This can be achieved by simply removing the low probability partitions. As noted earlier in Section 3.3.3, belief state hypotheses are formed from the combination of a partition, the last user action and the respective dialogue history. The probability of each hypothesis is maintained throughout the dialogue. The probability of a specific partition can be easily computed by marginalising out the user action and dialogue history by summing all hypotheses containing that partition *i.e.*, $b(p_t = p) = \sum_{h \in \mathcal{H}_t, h=(p,a_u,\mathbf{s}_d)} b(h_t = h)$. This allows for low probability partitions to be identified and removed. However, since the partitions are built in such way that they partition the whole space of user goals, completely removing a partition removes all the user goals that are represented by that partition. The remaining partitions do not represent all possible user goals anymore but just part of the goal space. This makes it impossible to recreate the removed user goals which is clearly not desirable.

Rather than removing the partitions, the method proposed in [110] reduces the number of partitions by recombining the low probability leaf partitions with their parent partitions. The recombination is performed by removing the complementary value from the parent partition, updating its probability with the probability of its child partition and removing the child partition.

Algorithm 5 Belief update with recombination

```
1:  $t \leftarrow 0$, Initialise $b(h_0)$
2:  repeat
3:    Partition each $p$ using attribute-value pairs from the last system action
      $a_m$
4:    Initialise $b(p_{t+1} = p) \leftarrow 0$ for all partitions $p$ in the current set of
      partitions
5:    for every user action $\tilde{a}'_u$ in N-best list $\mathbf{o}'$ do
6:      Partition each $p$ using attribute-value pairs from each $\tilde{a}'_u$, create
        new hypothesis $h'$ from each previous hypothesis $h$ associated with
        $p$ using $a_m$ and $\tilde{a}'_u$
7:      for all $p'$ in the current set of partitions do
8:        for each hypothesis $h'$ associated with $p'$ do
9:          Update belief $b(h_{t+1} = h')$ according to Eq. 3.4
10:         $b(p_{t+1} = p') \leftarrow b(p_{t+1} = p') + b(h_{t+1} = h')$
11:       end for
12:     end for
13:     Recombine partitions w.r.t the current updated belief $b(p_{t+1})$ so
        that a maximum of $\chi$ partitions remains
14:   end for
15:   Choose the next system action $a'_m$ according to $b(h_{t+1})$
16:   $t \leftarrow t + 1$
17: until dialogue ended
```

An outline of the belief update algorithm that implements this partition recombination is given in Algorithm 5. In each dialogue turn, the partitioning is performed using the attribute-value pairs from the last system action (line 3). Then, for each observation in the N-best user input the partitioning is performed using its attribute-value pairs (line 6), the belief over new hypotheses is updated (line 9) and the updated belief over partitions is accumulated (line 10). If the number of partitions exceeds the threshold, the partitions are recombined according to the current updated belief (line 13). After the whole N-best list is processed, the next system action is chosen according to the updated belief (line 15).

This method is shown to be effective in domains that do not have many attributes [110]. However, the method has its limitations in more complex domains. Firstly, a partition can only be recombined with its parent even though there may be other partitions it is complementary to and which would be better candidates for recombination. Referring to the example from Fig. 4.1-4.2, partition 1.1.1.1.1.1 is complementary both to partition 1.1.1.1.1 and to partition 1.1.1.1.2. Secondly, allowing only leaf partitions to be removed might not be desirable in long dialogues. Leaf partitions are usually the last to be created but in dialogues where the user goal evolves over time, the partitions that are created early on typically become less probable as the dialogue progresses. Thus, the more recent leaf partitions are often more useful. However, if one simply modifies the recombination technique to allow for non-leaf partitions to be recombined, as in for example 1.1.1 and 1.1.1.2 in Fig. 4.2, it becomes difficult to determine the right position for the newly obtained partition. What is more, such a partition would not have any complements so it would be impossible to remove it before other partitions are recombined. Finally, the problem of partitions without any complements can occur even in the case of recombining the leaf partitions. For example, recombining 1.1.1.1.1.1 with its parent 1.1.1.1.1 results in a partition that does not have any complements. In complex dialogues, where the user can change the goal, it may be important that each partition has a complementary partition.

In summary, the requirements for the pruning algorithm are the following. Firstly, the belief state after the pruning should be such that the system's error is minimal. Secondly, all possible user goals must remain represented. Thirdly, it must be always possible to perform the pruning. In the next section a description of the pruning algorithm that meets these requirements is given.

4.7.2 Pruning the applied attribute-value list

In this section, a new pruning method is presented that is not constrained by the position of partitions in the tree and guarantees that every partition has a complement.

Rather than recombining the partitions, the number of partitions can be reduced by removing some of the applied attribute-value pairs. The marginal probability of attribute-value pair $\alpha = \beta$ is the sum of probabilities of all partitions that have β set to true. In this way, a sorted list of the applied attribute-value pairs can be obtained. The lowest probability attribute-value pairs probably have the least impact on modelling the user goal and therefore can be removed.

Let $\alpha = \beta$ be the attribute-value pair with the lowest-probability in the list of applied attribute-value pairs. Assume that $\alpha = \beta$ is not among the superiors (see Section 4.3) for any other applied attribute-value pair $\alpha_k = \beta_k$. To remove $\alpha = \beta$, each partition that matches $\alpha = \beta$ has to be joined with its complementary partition that matches $\alpha = \neg\beta$. In order to find all such pairs of partitions the following procedure is taken. Starting from the root partition, partition p is examined to see if it contains node α with a child node η containing $\neg\beta$. If not, the search is continued through its children starting from the oldest. If node η does contain $\neg\beta$, partition p is marked as *upper*. Then the search is performed through the children of partition p, starting from the oldest, until one that contains node α with a child node η containing β is found. It is marked as *lower*. Such a pair of partitions is guaranteed to exist. The partitions are complementary and only differ in the atomic node η that contains β and for every given *upper* there is only one lower (see Stats. 1 and 5 in Appendix B.1). What is more, if partitions *upper* and *lower* have child partitions, they have subtrees of the same structure with these partitions as roots. Each partition from the *upper* subtree will have its complement in the *lower* subtree. All that is needed is to add the belief of each partition in the *lower* subtree to its complement in the *upper* subtree, to remove $\neg\beta$ from the *upper* partition and then to delete the *lower* subtree (see Stat. 6 in Appendix B.1). The procedure is applied repeatedly until there is no partition matching $\neg\beta$ left. This can be easily performed using a stack structure. The pseudo code is given in the Appendix B.2.

Every time a *lower* partition is removed all the hypotheses that are associated with that partition are removed. However, when a node is removed in an *upper* partition only the history related to that node is removed in each associated hypothesis.

An example of the pruning procedure is shown in Fig. 4.5, where the attribute-value pair food=Italian is removed from the list of applied attribute-value pairs. The first partition that contains node food and ¬Italian is 1.1.1 and its child partition that contains Italian is 1.1.1.2. They are respectively

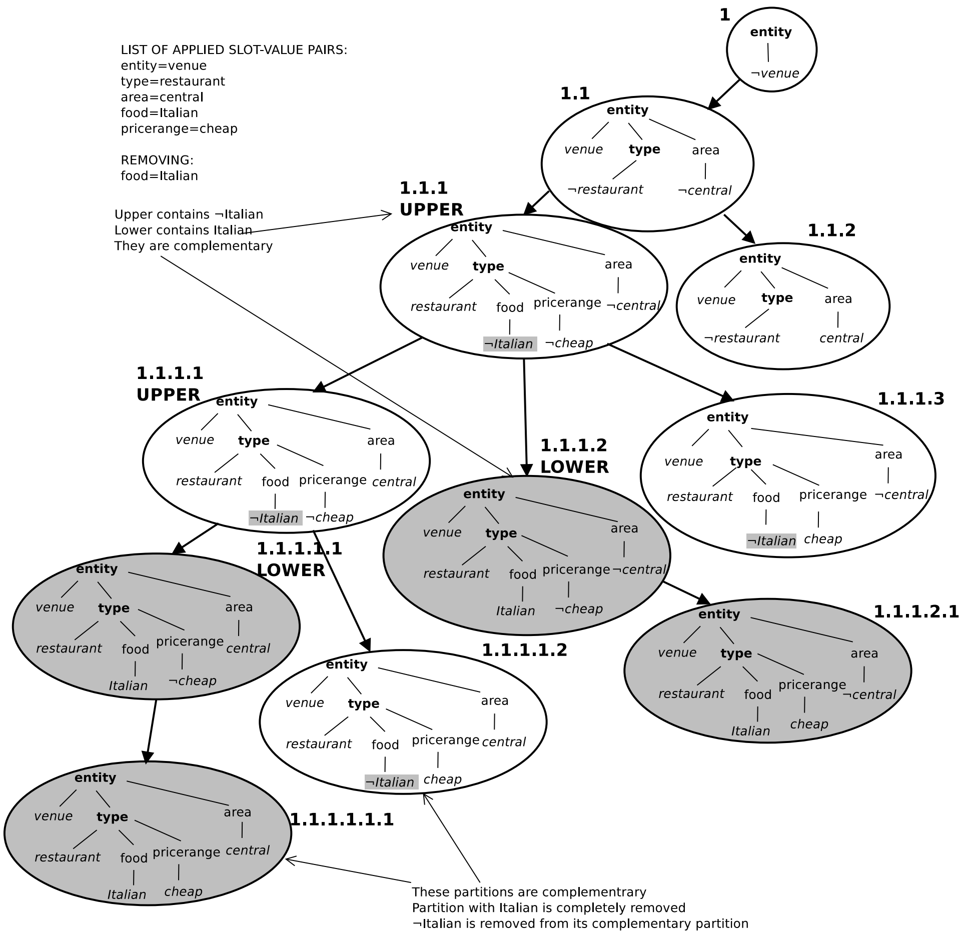


Figure 4.5: Pruning an attribute-value pair from the tree of partitions.

marked as *upper* and *lower* and both of them have child partitions which are complementary, 1.1.1.3 and 1.1.1.2.1 respectively, similarly for partitions 1.1.1.1 and 1.1.1.1.1. Then, partitions with Italian are deleted and ¬Italian is removed from their complementary partitions.

If $\alpha = \beta$ is among the superiors of some attribute-value pair $\alpha_k = \beta_k$, then $\alpha_k = \beta_k$ has to be pruned from the part of the tree that contains $\alpha = \beta$ (see Stat. 8 in Appendix B.1). The algorithm is the same as the one described above, with the difference that *upper* (3.4.2) and *lower* partitions are complementary in β_k and both contain β set to true. Referring to the example from Figs. 4.1-4.2, if, for instance, type=restaurant is to be removed, then food=Italian and area=central must be removed first. If the pruning is performed based on the lowest probability, it is never the case that food=Italian has higher probability than type=restaurant, since it can only occur in the partitions that have restaurant as true. However, some attributes can occur for different realisations of class nodes. For example, type=restaurant and type=hotel can both have pricerange=cheap (see Table 3.2). Because of this, if type=hotel is to be removed, pricerange=cheap only has to be removed from the partitions that have hotel.

Using this approach, one can guarantee that the lowest probability attribute-value pair can always be removed from the tree of partitions, regardless of when it was applied and how the partitions that contain it are structured in the tree. After pruning an attribute-value pair, the structure of the tree of partitions is the same as if that attribute-value pair had never been applied at all, so the pruning does not affect the existence of complements.

Whilst the partitioning process exponentially increases the number of partitions, the above pruning technique decreases it at the same rate, so the total number of partitions remains bounded. This allows dialogues of arbitrary length and it also enables large N-best inputs to be applied.

An outline of the belief update algorithm that applies the above pruning method is given in Algorithm 6. In contrast to Algorithm 5, pruning is applied before the processing of the N-best input, so that no information from the current N-best list is lost before the system action is chosen for the next turn. In order to accommodate negotiation-type dialogues, where the user can change their mind, more importance is placed on the most recent observations instead of keeping low probability, older information. In this representation, what the user does not want and what the system is

Algorithm 6 Belief update with pruning

```
1: $t \leftarrow 0$, Initialise $b(h_0)$
2: repeat
3:   for all applied attribute-value pairs $\alpha = \beta$ do
4:     $P(\alpha = \beta) \leftarrow \sum_{p \in \mathcal{P}_t, \alpha=\beta \in p} \sum_{h \in \mathcal{H}_t, h=(p,a_u,\mathbf{s}_d)} b(h_t = h)$
5:   end for
6:   Prune the list of the applied attribute-value pairs w.r.t. $P(\alpha = \beta)$ so
     that a maximum of $\chi$ partitions remains
7:   Partition each $p$ using attribute-value pairs from the last system action
     $a_m$
8:   for every user action $\tilde{a}'_u$ in N-best list $\mathbf{o}'$ do
9:     Partition each $p$ using attribute-value pairs from $\tilde{a}'_u$, create new
       hypothesis $h'$ from each previous hypothesis $h$ associated with $p$
       using $a_m$ and $\tilde{a}'_u$
10:    for all $p'$ in the current set of partitions do
11:      for each hypothesis $h'$ associated with $p'$ do
12:        Update belief $b(h_{t+1} = h')$ according to Eq. 3.4
13:      end for
14:    end for
15:  end for
16:  Choose system action $a'_m$ according to $b(h_{t+1})$
17:  $t \leftarrow t + 1$
18: until dialogue ended
```

uncertain about are represented in the same way to retain tractability. Low-probability slot-value pairs are the ones that the user is least likely to want so it is safer to remove them rather than the ones that were obtained from the most recent N-best input. At the beginning of each dialogue turn, the marginal probability of all applied attribute-value pairs is calculated (lines 3-5). Then, the lowest probability attribute-value pairs are pruned (line 6). Following this, the attribute-value pairs from the last system act are used for a new stage of partitioning (line 7). Next, partitioning is performed for each user dialogue act in the N-best input list, using its attribute-value pairs and the belief is updated (lines 8-15). Finally, the next system action is chosen based on the updated belief (line 16).

In Appendix C.2 an example of a long negotiation-type dialogue which incorporates this pruning technique is given together with the list of active attribute-value pairs and their marginal probabilities at each turn.

4.8 Evaluation results

The evaluation of the proposed partitioning techniques is divided into three parts. Section 4.8.1 examines how well the system can deal with user goals when the constraints are in the form of disjunctions. Then, in Section 4.8.2, the performance of the proposed pruning algorithm (described in Section 4.7.2) is compared to a version of the partition recombination algorithm [110]. Finally, in Section 4.8.3, the effect of different pruning thresholds on performance is examined.

For each experiment, the policy was trained and evaluated in interaction with the agenda-based simulated user as described in Section 3.4. The application is the Cambridge tourist information system mentioned in Section 4.1.1 and described in detail in Appendix C.1. In the set-up used here, the simulated user gives a reward at the end of each dialogue of 100 if the dialogue was successful and 0 otherwise, less the number of turns. This reward is slightly modified from the one used in Section 3.4.1 to encourage the system to engage in long dialogues where the user may change their mind. When the system makes an offer, the simulated user changes its goal with probability 0.143 (as determined by parameter ReqAltsAfterEntRec2 in Table 3.7 in Section 3.4.2). In the case when the system informs the user there is no matching entity the user always changes their mind.

The simulated user allows a maximum of 100 turns in each dialogue, terminating it when all the necessary information has been obtained or if the dialogue manager repeats the same dialogue action more than three times in a row. The Dirichlet error model (see Section 3.4.2) was used to add errors to the user input. In order to demonstrate the system's capability for dealing with reasonably long N-best lists, the length of the N-best list was set to 10. The system takes about 13 turns on average to complete a dialogue; Fig. 4.6 shows how the number of turns increases as the user input becomes more noisy. The policies were trained using the grid-based Monte Carlo control algorithm in an incremental noise setting (see Section 3.4). The resulting policies were evaluated with the simulated user performing 2500 dialogues at each error rate.

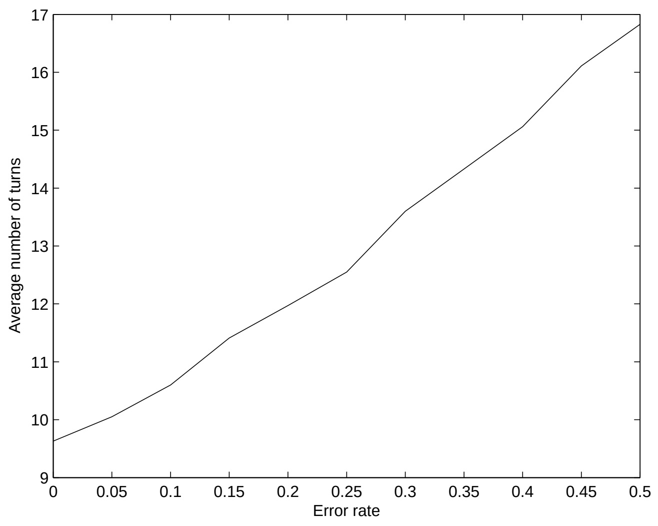


Figure 4.6: Average number of turns as a function of error rate.

4.8.1 Disjunctions in the user goal

In the first experiment, the simulated user was modified to produce constraints in the user goal such that on average 20% of them contain a disjunction of two values, for example, type=restaurant, pricerange = moderate $\vee$ cheap, food=Japanese. The performance of the system is compared on tasks that both contain and do not contain disjunctions. The performance is measured as the average reward at different confusion levels and the results are presented in Fig. 4.7[1]. As can be seen from the graph, the system can deal with disjunctions in the user constraints at least as well as it can for standard user constraints.

4.8.2 Pruning vs recombination

The second experiment compares the two methods for reducing the number of partitions: pruning the applied attribute-value list and the partition recombination algorithm. In both cases, the maximum number of active partitions χ was set to 300. The performance was examined by measuring the average reward that the system obtained with each of the methods. The results are given in Fig. 4.8, which shows that pruning the applied attribute-

[1]The error bars represent a 95% confidence interval.

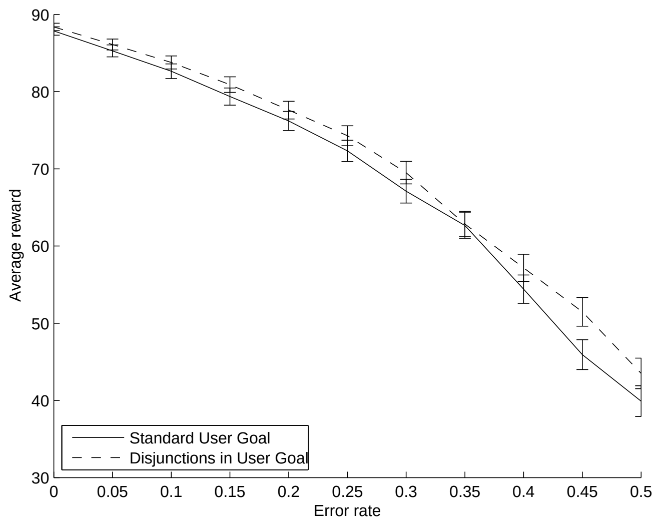


Figure 4.7: System performance with disjunctions in the user's request.

value list gives a better overall performance. As shown by the error bars, the results are statistically significant in the high noise regions, suggesting that it can more effectively manage user goal partitioning in noisy complex domains compared to the simpler partition recombination approach.

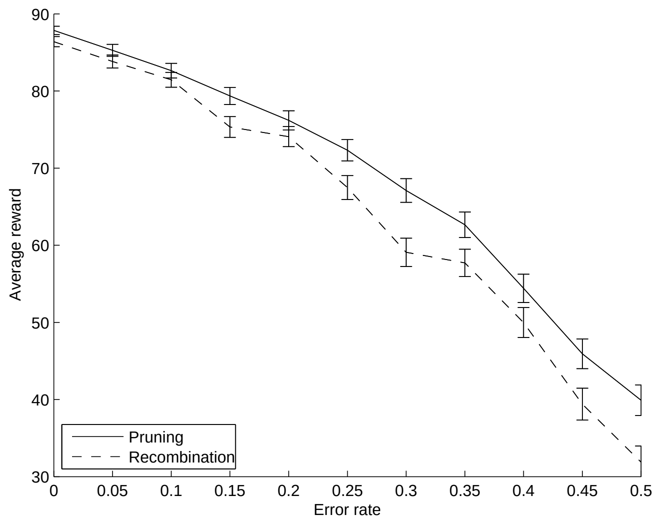


Figure 4.8: System performance comparison of attribute-value list pruning (Pruning) and partition recombination (Recombination).

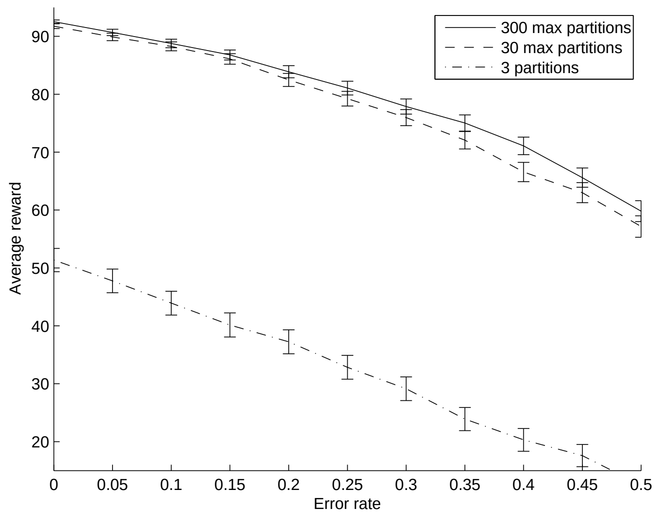
4.8.3 The effect of different pruning thresholds

In order to examine the effect that the pruning has on the system's performance, three different pruning thresholds χ were compared: 3, 30 and 300. In addition, a contrast is provided between two different user simulator settings – one where the user goal stays constant during the dialogue, and one where it changes. The performance comparison is given in Fig. 4.9.

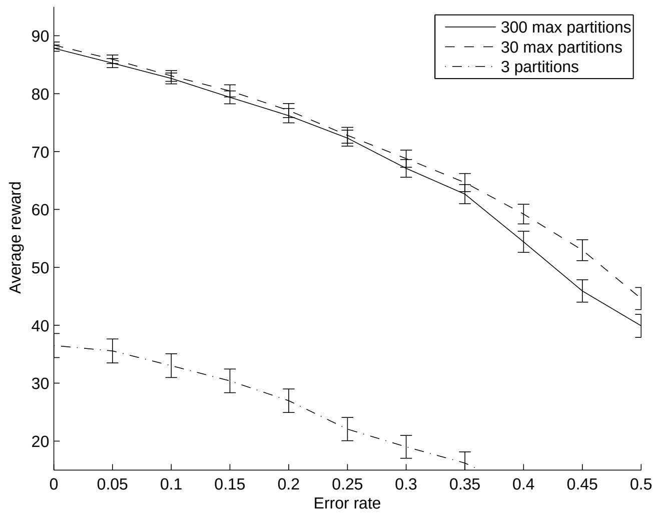
In the case when the user goal stays constant during the dialogue, increasing the number of partitions leads to improved system performance. This is in line with the findings in [110]. It is important to note that, in contrast to the dramatic difference in performance between pruning threshold 3 and 30, the difference between 30 and 300 is largely not statistically significant. 3 partitions roughly correspond to only 2 attribute-value pairs representing the user goal and the user typically has about 4 constraints, so the goal in this case is not fully represented. Therefore, if the dialogue manager has a very low pruning threshold it is not able to represent the user goal even when there are no errors, which leads to low performance. The results from Fig. 4.9 suggest that increasing the number of partitions over 300 would not improve the performance further and this was confirmed by further tests with a threshold of 3000. In the case where the user goal changes during the dialogue, the threshold of 30 gives a more robust performance on higher error rates then the threshold of 300, see Fig. 4.9(b). This is probably a consequence of the fact that the HIS system does not have an explicit state transition matrix. A change of user goal can also be achieved by discarding earlier evidence in favour of the most recent evidence, and pruning helps achieve this. Thus, in the HIS model, pruning enables the dialogue to be more robust to inconsistent user behaviour. In real dialogues, users do not normally have a strictly defined goal but are likely to change their mind depending on the system's response, and pruning can facilitate this.

4.9 Summary

This chapter has described how enriching the dialogue state structure with an explicit representation of complements can improve POMDP-based dialogue modelling in a complex domain. The proposed representation enables the use of disjunctions and conjunctions in the user request as well as quanti-



(a) User goal stays constant during the dialogue



(b) User goal changes during the dialogue

Figure 4.9: Influence of the pruning threshold on system's performance when the user has a constant goal (a) and when the user's goal changes (b).

fiers in the system's response. More importantly, the notion of complements provides a basis for a pruning technique that can effectively bound the number of partitions created during a dialogue and thereby ensure tractability. It supports N-best lists of user dialogue act hypotheses which are large enough to include all of the informative hypothesised utterances from noisy speech and it can handle dialogues of arbitrary length. It was shown that this new pruning technique leads to better performance than an existing recombination method in a practical real-world application domain. The next chapter discusses how policy optimisation can further improve the performance of POMDP-based dialogue managers.

Chapter 5

Policy optimisation in summary space

5.1 Introduction

Policy optimisation in a POMDP-based dialogue system faces many challenges. Many of these are the result of the approximations required to achieve tractability in the learning process. Section 3.3.5 showed that for real-world systems, one must compress the master belief space into a much smaller summary space for learning to be tractable. However, learning within a summary space is not trivial. When using a summary space, many belief points will map to the same summary point for which the policy proposes a summary action. That summary action might not always be applicable for each of the original belief points, and there is then a problem in performing the inverse mapping back to the master space.

This chapter discusses the invalid action problem in more detail. Section 5.4 proposes an extension to the original grid-based Monte Carlo control algorithm to deal with the invalid action problem. Results and different contrasts are then presented in Section 5.5.

5.2 The invalid action problem in a POMDP

In a partially observable Markov decision process, the belief state represents a probability over all states. Since any state is possible in any belief state, the policy must include the possibility of taking any action in any belief

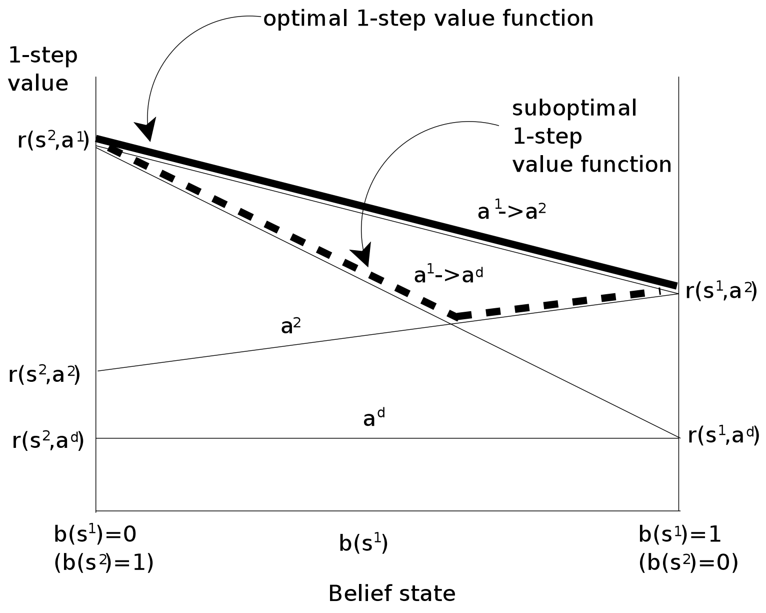


Figure 5.1: Two state POMDP with an impossible action-state pair.

state. However, if an action is not valid in some states, every belief state will have a non-negative probability that the action is invalid. This poses a problem for policy optimisation.

Under the assumption that once an action is proposed it is possible to determine whether or not it is valid, even if the actual state remains unknown, this problem can be dealt with in different ways. One way would be to expand the action set so that a single default action which is guaranteed to be valid in any state is associated with every action. However, this can lead to suboptimal results.

Consider, for example, a simple POMDP with state space $\mathcal{S} = \{s^1, s^2\}$, some finite observation space, and action space $\mathcal{A} = \{a^1, a^2, a^d\}$, where action a^1 is not valid in state s^1. The rewards are $r(s^1, a^2) = 3$, $r(s^1, a^d) = 1$, $r(s^2, a^1) = 5$, $r(s^2, a^2) = 2$ and $r(s^2, a^d) = 1$. In order to perform policy optimisation, each action has to be applicable for any belief state $b(s_t)$. In order to allow action a^1 to be performed in any belief state $b(s_t)$, it has to be given a valid interpretation in state s^1. This can be done by redefining action a^1 as $a^1 \to a^d$, meaning that if action a^1 is not valid, back-off to default action a^d instead. In this way, this new action $a^1 \to a^d$ is used for policy optimisation instead of action a^1. After applying 1-step of the

POMDP Value Iteration algorithm (Section 2.2.5), there are three different policy trees: one associated with action a^d, one associated with action a^2 and one associated with action $a^1 \rightarrow a^d$. The optimal 1-step Value function therefore comprises $a^1 \rightarrow a^d$ and a^2 (dashed upper surface in Fig. 5.1). However, this is clearly a suboptimal solution compared to $a^1 \rightarrow a^2$ (solid line in Fig. 5.1). Had action a_1 been replaced with actions $a^1 \rightarrow a^d$ and $a^1 \rightarrow a^2$ there would be four policy trees and 1-step Value iteration would yield $a^1 \rightarrow a^2$ as the optimal 1-step Value function. This demonstrates that every possible back-off strategy should be considered, as it could be a part of the optimal policy.

In the general case, the optimal back-off for action a^i will be a sequence $a^i \rightarrow a^{i_1} \ldots \rightarrow a^{i_j} \ldots \rightarrow a^{i_{|\mathcal{A}|-1}}$. Redefining each action to include all possible back-off actions would increase the cardinality of the action set to $n!$ and this is clearly not feasible.

An alternative way of finding the optimal back-off action is to use the Q-values. The Q-value, $Q(\mathbf{b}, a)$, is the expected reward from taking action a in belief state $\mathbf{b}$ and following the policy thereafter. It can be calculated as $Q(\mathbf{b}, a) = \sum_{s \in S} Q(s, a) b(s_t = s)$, where $Q(s, a)$ is the Q-value of taking action a in state s (see Eq. 2.14 in Section 2.2.5). For each action a, the set of states in which that action can be taken is known. The learning algorithm can estimate $Q(\mathbf{b}, a)$ by summing over only the $Q(s, a)$ values for which action a is valid in s. Ordering $Q(\mathbf{b}, a)$ in a list for each action a provides a sequence of possible back-off actions which can be searched until a valid action is found in belief state $\mathbf{b}$. This intuition is the motivation behind the N-best back-off action selection mechanism described further in this chapter.

5.3 The invalid action problem in a summary space-based POMDP dialogue manager

In a dialogue system, given some prior information on user preferences, there is nothing in principle to stop the system from saying as its very first action: "*Please confirm that you want a Chinese restaurant?*", even though the user has not yet said anything. However, the invalid action problem presents itself in a system which optimises the policy in a compressed, summary space, choosing between actions which are simple strategic decisions such as "inform", "confirm", etc. In this case, the fact that an action such as "confirm" is not valid in a particular belief state cannot be determined from the summary state itself since many different belief states may map to the same summary state. Only after mapping back to the master state, when the information from most likely hypothesis has to be taken to form the system dialogue act, will it be discovered that there is no information which can be confirmed.

The discretisation of the summary space further exacerbates the invalid action problem. The effect of discretisation is to represent all belief points in a neighbourhood by a single representative grid point. Even if there are no visited summary points within the neighbourhood with invalid actions, merging them into a single grid region will generate the union of actions for the whole region and this union might have invalid actions. One possibility to avoid this is to increase the information transferred from master to summary space with the subset of actions that are valid in that summary state. This is further examined in the experimental section below.

An alternative approach is to amend the grid-based learning with back-off action selection that uses Q-function values. A detailed algorithm for implementing this is given in the next section.

5.4 The extended grid-based Monte Carlo control algorithm

In this section, an extension to the Monte Carlo control algorithm (Algorithm 2, Section 2.2.1) is given that ranks the alternative actions for each point in belief space into an N-best list ordered by Q-value. The basic idea is simple. The POMDP summary space is represented by a number of discrete grid points. The system interacts with a user and Q-values are estimated for each grid point and each action. The action with the highest Q-value at each grid point then forms the policy. Extending the standard algorithm to include N-best action selection is simply a matter of storing a rank-ordered list instead of only the highest Q-value at each grid point.

Learning starts by arbitrarily assigning values to the Q-values for each action a associated with the initial grid point $\hat{\mathbf{b}}^0$. In addition to the Q-values, an N-value is associated with each grid point-action pair $(\hat{\mathbf{b}}, a)$. The N-value records the number of times that action a is taken in grid point $\hat{\mathbf{b}}$.

Algorithm 7 Extended grid-based Monte Carlo control algorithm

```
1: $\hat{\mathcal{B}} = \{\hat{\mathbf{b}}^0\}$ initial grid point
2: for all $a \in \mathcal{A}$ do
3:     $Q(\hat{\mathbf{b}}^0, a) \leftarrow$ arbitrary
4:     $N(\hat{\mathbf{b}}^0, a) \leftarrow$ arbitrary
5: end for
6: $\pi(\hat{\mathbf{b}}^0) = a^{i_1} \dots a^{i_{|\mathcal{A}|}}$ arbitrary order
7: repeat
8:     repeat
9:         Update current belief state $\mathbf{b}$
10:        $\hat{\mathbf{b}} \leftarrow$ GridPoint(SummaryState($\mathbf{b}$))
11:        $a^{i_1}, \dots, a^{i_{|\mathcal{A}|}} \leftarrow \begin{cases} \text{arbitrary order} & \text{with prob. } \epsilon \\ \pi(\text{Nearest}(\hat{\mathbf{b}}, \hat{\mathcal{B}})) & \text{with prob. } 1-\epsilon \end{cases}$
12:        record $(\hat{\mathbf{b}}, a)$, where $a$ is the taken action
13:    until episode terminates
14:    for pairs $(\hat{\mathbf{b}}, a)$ appearing in the episode do
15:        $R \leftarrow$ discounted return following the occurrence of $(\hat{\mathbf{b}}, a)$
16:        if $\exists \hat{\mathbf{b}}^k \in \hat{\mathcal{B}}$, $|\hat{\mathbf{b}} - \hat{\mathbf{b}}^k| < \nu$ then
17:            $Q(\hat{\mathbf{b}}^k, a) \leftarrow \frac{Q(\hat{\mathbf{b}}^k,a)*N(\hat{\mathbf{b}}^k,a)+R}{N(\hat{\mathbf{b}}^k,a)+1}$
18:            $N(\hat{\mathbf{b}}^k, a) \leftarrow N(\hat{\mathbf{b}}^k, a) + 1$
19:        else
20:            add $\hat{\mathbf{b}}$ to $\hat{\mathcal{B}}$, $Q(\hat{\mathbf{b}}, a) \leftarrow R$, $N(\hat{\mathbf{b}}, a) \leftarrow 1$
21:        end if
22:    end for
23:    for all $\hat{\mathbf{b}}^k \in \hat{\mathcal{B}}$ do
24:        $\pi(\hat{\mathbf{b}}^k) =$ actions ordered by $Q(\hat{\mathbf{b}}^k, a)$
25:    end for
26: until converged
```

Each learning episode is conducted ϵ-greedily *i.e.,* the current best policy ($\pi(\hat{\mathbf{b}}) = \arg\max_a Q(\hat{\mathbf{b}}, a)$) is used with probability $1-\epsilon$ and with probability ϵ a random action is taken. In the standard algorithm, the policy consists of one action per grid point – this is the action that has the highest Q-value. For N-best action selection, the algorithm is modified so that the policy consists of a list of actions per grid point, ordered by their Q-values. In a similar way, when exploring, instead of generating one random action, a random ordering of actions is generated. In all cases, the reward obtained for each turn is assigned to the Q-value for the action that was actually taken. Every time a new belief state $\mathbf{b}$ is visited, it is mapped to a summary state $\hat{\mathbf{b}}$ and then to the nearest grid point in the set of grid points $\hat{\mathcal{B}}$. If there is no nearby grid point, a new one is created and added to $\hat{\mathcal{B}}$. Thus, during training grid points are created with their respective lists of Q- and N-values. A complete description is given in Algorithm 7.

5.5 Evaluation results on simulated user

This section presents results for the proposed N-best back-off action selection method and compares it with a simple fixed back-off baseline and an alternative based on extending the summary space with features designed to minimise the occurrence of invalid actions. For each experiment, the policy was trained and evaluated in interaction with the agenda-based simulated user as described in Section 3.4.

5.5.1 Fixed back-off

The simplest solution to the back-off problem is to use a single global default action as the back-off action. The requirement for this action is that it should be available at any point in the belief space. In the HIS system, the only appropriate candidates are the UserRepeat action where the system asks the user to repeat the last input, or the Bye action where the system ends the dialogue, (see Table 3.6). In the case of UserRepeat, users typically repeat the last dialogue act or hang up if the system has already asked this several times. By asking the user to repeat the last act, the system can potentially obtain a better estimate of the current user state, but it can also waste time, leading to a lower reward. The Bye action would only be appropriate for a system which could divert to an operator, otherwise its use would be unacceptable. Hence, the UserRepeat action is used for the fixed back-off baseline. The training schedule adopted was the same as in Section 3.4.3. The performance is shown in Figs. 5.2 and 5.3.

5.5.2 Extension of the summary space

The second approach to the problem of invalid actions is to extend the summary space with explicit information about which action can be taken. In this approach, each summary grid point is augmented with a set of binary flags. Flags are defined for each summary action and indicate whether or not the action is valid for that grid point. If two belief states have different subsets of plausible actions then they are mapped to different grid points. Since there are 11 possible summary actions (see Table 3.6 in Section 3.3.5), this can potentially increase the size of the summary space by a factor of 2^{11}.

In practice, however, some actions are always possible and there are also dependencies between them. In the experiments conducted here, the extended summary space resulted in a policy with 2000 grid points. The training scheme used for this system was similar to the the one in Section 3.4.3, with the difference that training had ten times more dialogues to compensate for the increased number of grid points. The performance of this system is shown in Figs. 5.2 and 5.3.

There is a third approach that could be taken to deal with the problem of invalid actions. For a given summary point, one can find the closest grid point which has a valid action as its choice. The corresponding action is then taken. While this approach avoids hand-crafting and extending the summary space, the problem is that it approximates a summary point with a grid point that is further away than the closest one. This increases the area of summary space used to estimate the Q value, which makes the estimates less accurate. Worse than this, the area of belief space used to estimate the Q-values for a summary point will vary over time and between actions. This allows for a potential bias, increases the variance in the estimates and could cause serious issues during training.

As an example, consider a sample belief point $\mathbf{b}$. Due to insufficient training, the current estimate of the top action for the closest summary point $\hat{\mathbf{b}}^i$ may not be valid, even though the optimal Q-function would yield a valid top action. The strategy then finds the closest summary point $\hat{\mathbf{b}}^j$

having a valid top action and assigns the achieved reward to its Q-value. Later in the training, the action will become valid but by that stage the reward is already included in estimating the Q-value of $\hat{\mathbf{b}}^j$. That Q-value is therefore biased by including rewards for points that will not be seen again, and at the same time the Q-value at $\hat{\mathbf{b}}^i$ is less accurate, leading to slower convergence. On the other hand, if the reward is assigned to the Q-value of the closest point $\hat{\mathbf{b}}^i$, there is a mismatch between the Q values used to make decisions and the estimated Q-values, which violates the requirements of the algorithm. If the rewards are to be assigned in this way, then one must use the ranking obtained by the valid actions' Q-values at $\mathbf{b}$. This gives a fourth strategy, which is the N-best strategy examined here.

5.5.3 N-best back-off

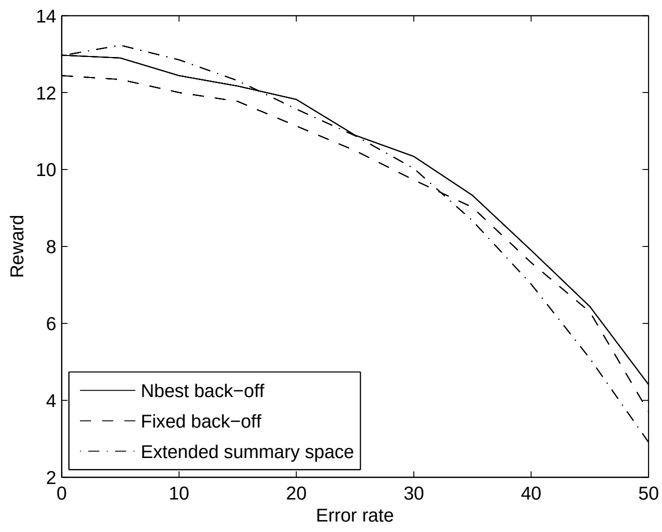


Figure 5.2: Average reward for simulated dialogues between the fixed back-off strategy, the strategy with extended summary space and the N-best back-off strategy on different error rates.

Q-function values from the Monte Carlo control algorithm ranks a list of actions by policy preference associated with each summary state (see Section 5.4). Using this N-best list for back-off action selection, the policy was trained using the same training scheme as in the fixed back-off strategy. The performance results for this system are given in Figs. 5.2 and 5.3.

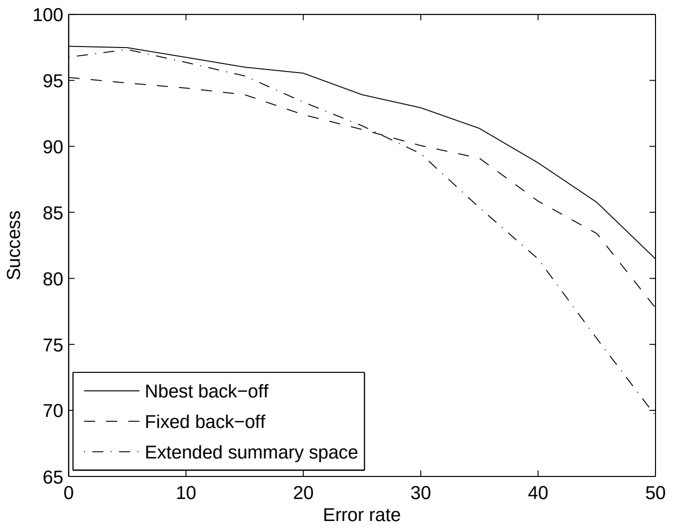


Figure 5.3: Comparison of the percentage of the successfully completed simulated dialogues between the fixed back-off strategy, the strategy with extended summary space and the N-best back-off strategy on different error rates.

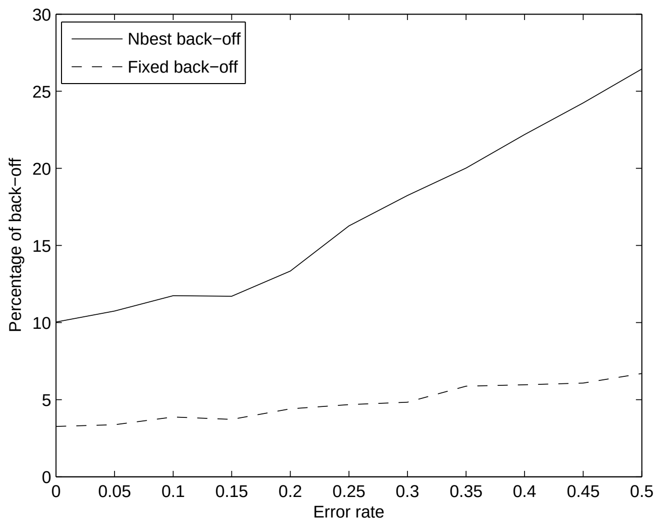


Figure 5.4: Percentage of backed-off actions for the Fixed back-off strategy and the N-best back-off strategy for different error rates.

Average rewards are relatively close to each other for all of these strategies (Fig. 5.2). However, the average success in Fig. 5.3, shows that the N-best back-off outperforms both alternative strategies across all error rates. The extension of the summary space improved the performance on low error rates. However, because the summary space was extended, the performance degraded rapidly in noise and even increasing the number of training dialogues by a factor of 10 was not able to compensate for this.

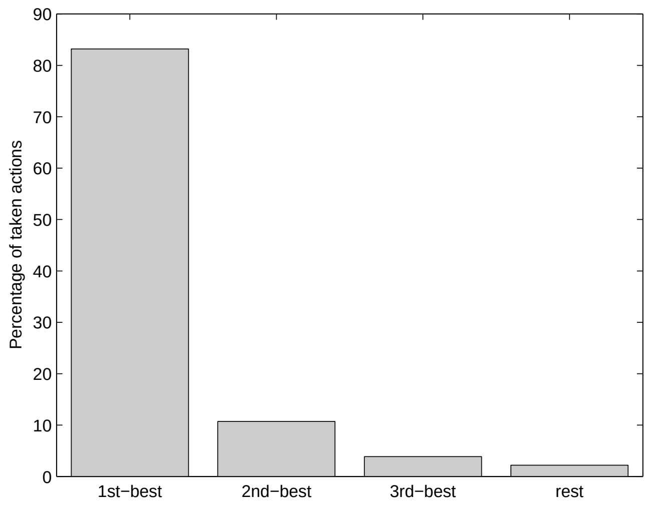


Figure 5.5: Percentage of actions on different positions in the N-best list taken in the N-best back-off strategy.

It is also interesting to note that the frequency with which the top proposed action is not taken differs in the N-best and fixed back-off strategies. The policy obtained using the N-Best back-off strategy backs off more often (see Fig. 5.4). This suggests that this policy has more liberty when choosing its action, since there is a list of back-off actions to try if the top action fails. Not only does the N-best strategy back off more, but the back-off rate increases more dramatically with noise than the fixed back-off strategy. This can be ascribed to the difficulty of correctly determining which actions are valid in noisy states. The N-best strategy tries the actions that, if valid, achieve the best reward, whereas the fixed back-off chooses the actions that rarely lead to back-off. Fig. 5.5 shows the percentage of the first-, the second- and the third-best action taken. The results were obtained from 2500 dialogues for each error rate. It shows that on average, 82% of the time, the top action can be mapped to a master action, but there is a significant tail when the system backs off to the second- and the third-best action.

5.6 Summary

Invalid state-action pairs are an intrinsic feature of summary-based methods for POMDPs. Available solutions try to back off to a default action that is defined everywhere, but this leads to sub-optimal performance. Alternative solutions to this problem have been examined in the framework of a real-world summary space-based POMDP dialogue system. In a baseline back-off strategy, one summary action is chosen to be the back-off action. In another strategy, the summary space was extended to include information about the subset of actions which are plausible in that particular state. However, this resulted in fragmentation of the space, increased demand for training dialogues and poor robustness to noise.

On the other hand, the strategy of associating an N-best list of actions ranked by Q-value with each grid point worked well. This approach provided the best performance at all error rates and, importantly, it did not need any more training data than the fixed back-off strategy.

Apart from showing how dialogue performance can be improved by the correct choice of back-off actions, these results show the inability of grid-based reinforcement algorithms to deal with the extension of the summary space, leading to reduced robustness. This is a major constraint since it places a limit on which parts of the system's behaviour can be automatically optimised. This motivates the need for an alternative policy optimisation approach that can facilitate faster learning, which will be investigated in the next chapter.

Chapter 6

Gaussian processes for fast policy optimisation

6.1 Introduction

The POMDP policy optimisation procedure described previously uses grid-based approximations of the corresponding continuous state MDP. This allows the use of a discrete MDP reinforcement learning algorithm to optimise the Q-function at the grid points, which then yields an optimised policy. Section 3.4 showed that in order to learn a Q-function for a real-world dialogue task with approximately 700 grid points, around $100,000$ dialogues are required. This number is too large for the optimisation to take place in interaction with real users so the interaction is normally performed with a simulated user. This is not ideal because of potential discrepancies between real and simulated user behaviour. At the same time, the number of grid points is often relatively small, which brings the quality of the approximation into question. Unfortunately, further extension of the summary space was shown in Section 5.5 to lead to a loss in performance. The main cause of this deficiency is that an estimate of the Q-function at one grid point does not contribute to the estimate at another grid point. No matter how similar the behaviour of the Q-function is at these points, the values are always considered independent. Therefore, the grid-based approach has a slow convergence rate and limits the ability to extend the summary space, which is clearly not desirable. An alternative algorithm which does not have these problems would be an important improvement.

A common way to deal with MDPs with continuous state and action spaces is to use a parametric approximation of the Q-function or the policy. This is normally represented by a combination of basis functions [32, 73]. This approach is very effective at ensuring tractability and removes the need for storing large tables of grid-points. Instead, it only requires that a relatively small number of parameters be stored. A drawback, however, is that the solution is only optimal within the given basis. In particular, for dialogue policy optimisation, it is often difficult to know which basis function would represent the Q-function appropriately and this normally requires that the set of feature functions are defined using expert knowledge about the domain. This chapter proposes an alternative approach which does not make such strong assumptions about the form of the Q-functions. Instead, a non-parametric approach based on Gaussian processes (GPs) is used to overcome the drawbacks of these alternative approaches. Note that a non-parametric approach does not mean a parameter-free approach, but rather that regardless of how the model parameters are set, given enough samples the model converges to the optimal solution [118].

Gaussian processes are non-parametric Bayesian models used for function approximation. They have been successfully applied to reinforcement learning for continuous-space MDPs, using both model-free approaches [119, 120] and model-based approaches [118, 121]. In this chapter, the Gaussian process is used as a model for the Q-function. An advantage of Bayesian approaches is that they offer a principled way of incorporating prior knowledge about the underlying task in the learning process, which gives them the potential to improve learning rates. As already noted, it is important that the dependencies of the Q-function in the different parts of the belief state space are taken into consideration during learning. Gaussian processes are able to incorporate the prior knowledge of these dependencies elegantly through the choice of the *kernel function*, the purpose of which is to describe correlations in different parts of the space. In an ideal case, this approach would fully replace the need for a summary space by modelling correlations directly in the full belief space, although it is possible to apply them on the summary space as well. In addition, Gaussian processes allow the variance of the posterior to be estimated, thus modelling the uncertainty of the Q-function approximation. This is particularly useful for dialogue management, as for every belief state-action pair the Gaussian process does not only provide the

Q-function estimate, but also a measure of uncertainty in taking that action in that belief state. This potentially allows the estimate that was obtained through interaction with the simulated user to be further refined in interaction with real users. The application of Gaussian processes to model-free reinforcement learning is therefore well suited to dialogue management.

The next section presents an overview of Gaussian processes for model-free reinforcement learning. In Section 6.3 the core idea of the method is explained on a toy dialogue problem, where different aspects of GP learning are examined and the results are compared. Section 6.4 then demonstrates how this methodology can be applied to the HIS dialogue manager for a real-world problem.

6.2 Gaussian process reinforcement learning

This section provides an overview of Gaussian processes in reinforcement learning. Gaussian process regression is reviewed [37], and a method for extending the approach to reinforcement learning is described. Particular aspects of Gaussian process reinforcement learning that are discussed are the choice of kernel function, the computational complexity and efficient exploration of the state space. A Gaussian process reinforcement learning algorithm is described and its advantages are discussed in the light of dialogue management.

6.2.1 Gaussian process regression

A Gaussian process is a set of random variables, any finite subset of which is jointly Gaussian distributed. A Gaussian process, denoted as $f(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x})\right)$, is fully specified by its mean function $m(\mathbf{x}) = E\left(f(\mathbf{x})\right)$ and its covariance function $k(\mathbf{x}, \mathbf{x}') = E\left((f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))\right)$, also called the kernel function. The kernel function is a positive definite function that represents function correlations. It defines how the function values in two points are correlated. The positive definiteness ensures that any multivariate Gaussian distribution derived from the Gaussian process is well-defined.

When using the Gaussian process, one starts by specifying the prior and the aim is then to compute a posterior given some observations. In the simplest case of GP regression, $f(\mathbf{x})$ is the unknown function for which an approximation is sought, and a zero-mean Gaussian process with kernel function $k(\mathbf{x}, \mathbf{x}')$ is assumed as the prior distribution, $f(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}))$. The posterior distribution of $f(\mathbf{x})$ given the function values at some representative data points could then be computed using simple algebraic formulae [37].

In most real-world applications however, the true value of the function at the data points is not available, and only some noisy observations may be obtained. Let $\mathbf{y}_t = [y_1, \ldots, y_t]^\mathsf{T}$ be noisy observations of the function at data points $\mathbf{X}_t = [\mathbf{x}_1, \ldots, \mathbf{x}_T]^\mathsf{T}$, and $y_i = f(\mathbf{x}_i) + \xi$ is assumed, where the noise is additive, independent and Gaussian distributed, $\xi \sim \mathcal{N}(0, \sigma^2)$. For a Gaussian process, the joint distribution of the observed values and the function value at a test point, $f(\mathbf{x})$, can be shown to be [37]:

$$\left[\begin{array}{c} \mathbf{y}_t \\ f(\mathbf{x}) \end{array}\right] \sim \mathcal{N}\left(\mathbf{0}, \left[\begin{array}{cc} \mathbf{K}_t + \sigma^2 I & \mathbf{k}_t(\mathbf{x}) \\ \mathbf{k}_t^\mathsf{T}(\mathbf{x}) & k(\mathbf{x}, \mathbf{x}) \end{array}\right]\right), \tag{6.1}$$

where $\mathbf{K}_t$ is the Gram matrix, *i.e.,* the matrix of kernel function values between each pair of data points, $(\mathbf{K}_t)_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, where $i, j \in \{1, \ldots, t\}$, and $\mathbf{k}_t(\mathbf{x})$ is the vector of kernel function values between test point $\mathbf{x}$ and each data point, $\mathbf{k}_t(\mathbf{x}) = [k(\mathbf{x}, \mathbf{x}_1) \ldots k(\mathbf{x}, \mathbf{x}_t)]^\mathsf{T}$.

Conditioning the joint Gaussian prior (Eq. 6.1) on the observations yields the posterior of $f(\mathbf{x})$ [37]:

$$\begin{aligned} f(\mathbf{x})|\mathbf{y}_t &\sim \mathcal{N}\left(\overline{f}(\mathbf{x}), cov(\mathbf{x}, \mathbf{x})\right), \\ \overline{f}(\mathbf{x}) &= \mathbf{k}_t(\mathbf{x})^\mathsf{T}(\mathbf{K}_t + \sigma^2 I)^{-1}\mathbf{y}_t, \\ cov(\mathbf{x}, \mathbf{x}) &= k(\mathbf{x}, \mathbf{x}) - \mathbf{k}_t(\mathbf{x})^\mathsf{T}(\mathbf{K}_t + \sigma^2 I)^{-1}\mathbf{k}_t(\mathbf{x}). \end{aligned} \tag{6.2}$$

From Eq. 6.2 it is required that data points $\mathbf{X}_t$ are known since $\mathbf{y}_t$ are observations at these points. In the literature this is often emphasised by the notation $f(\mathbf{x})|\mathbf{y}_t, \mathbf{X}_t$. From that notation, it is clear that the posterior depends on data points and the observations at the data points. It is important to note, however, that the variance of the posterior only depends on the data points and not the observations. With more observations, the variance decreases and the estimate of the mean is refined.

6.2.2 The Q-function as a Gaussian process

As outlined in the introduction, a Gaussian process can be used to model the Q-function in reinforcement learning. The approach that is adopted here is a model-free approach, based on the description given in [120].

A discrete-space POMDP can be viewed as a continuous-space MDP, see Section 2.2.6. If $\mathcal{S}$ is the POMDP state space, its belief state at turn t, $b(s_t)$, can be seen as the state variable in its corresponding MDP. It takes values $\mathbf{b} \in \mathcal{B}$, where $\mathcal{B}$ is a continuous space of dimensionality $|\mathcal{S}|$, namely $[0,1]^{|\mathcal{S}|}$. In dialogue management, however, this space may be reduced to a summary space that contains both continuous and discrete variables, see Section 3.3.5. Therefore, in the most general case, the approximation framework needs to support modelling of the Q-function in a multidimensional space that consists of both continuous and discrete variables. A Gaussian process allows such modelling. For now, the MDP with the full belief state space $\mathcal{B}$ is considered, but it will be shown later how the description generalises to summary spaces.

As defined by Eq. 2.1 in Section 2.2.1, the discounted return R_t^π for time step t and a given policy π is the total accumulated reward acquired using that policy starting from that time step:

$$R_t^\pi = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}, \tag{6.3}$$

where r_{t+i+1} is the immediate reward in time step $t+i+1$ and γ, $0 < \gamma < 1$, is the discount factor. If the immediate reward is a random process, the discounted return is also a random process.

The discounted return for policy π can be written recursively as:

$$R_t^\pi = r_{t+1} + \gamma R_{t+1}^\pi. \tag{6.4}$$

The Q-function for policy π is the expectation of the discounted return for that policy given belief state $\mathbf{b}$ and action a in time t, over all possible belief state sequences that can be generated with policy π (see Eq 2.3 in Section 2.2.1):

$$Q^\pi(\mathbf{b}, a) = E_\pi \left(R_t | b(s_t) = \mathbf{b}, a_t = a \right). \tag{6.5}$$

If the transition probabilities do not change over time, $Q^\pi(\mathbf{b}, a)$ is not a random value. However, during the process of estimation it can be considered to be a random variable.

The discounted return can therefore be decomposed into a mean $Q^\pi(\mathbf{b}, a)$

and a residual $\Delta Q^{\pi}(\mathbf{b}, a)$:

$$R_t^{\pi}(b(s_t) = \mathbf{b}, a_t = a) = Q^{\pi}(\mathbf{b}, a) + \Delta Q^{\pi}(\mathbf{b}, a). \tag{6.6}$$

Substituting R_t^{π} and R_{t+1}^{π} from Eq. 6.6 into Eq. 6.4 yields:

$$r_{t+1}(b(s_t) = \mathbf{b}, a_t = a) = Q^{\pi}(\mathbf{b}, a) - \gamma Q^{\pi}(\mathbf{b}', a') + \Delta Q^{\pi}(\mathbf{b}, a) - \gamma \Delta Q^{\pi}(\mathbf{b}', a'), \tag{6.7}$$

where $b(s_{t+1}) = \mathbf{b}'$ is the next belief state and $a' = \pi(\mathbf{b}')$ is the next action, $a_{t+1} = a'$.

Let $\mathbf{B}_t = [(\mathbf{b}^0, a^0) \dots, (\mathbf{b}^t, a^t)]^{\mathsf{T}}$ be a sequence of t belief state and action samples[1] generated with policy π. Then, Eq. 6.7 becomes:

$$\begin{aligned}
r^1 &= Q^{\pi}(\mathbf{b}^0, a^0) - \gamma Q^{\pi}(\mathbf{b}^1, a^1) + \Delta Q^{\pi}(\mathbf{b}^0, a^0) - \gamma \Delta Q^{\pi}(\mathbf{b}^1, a^1) \\
r^2 &= Q^{\pi}(\mathbf{b}^1, a^1) - \gamma Q^{\pi}(\mathbf{b}^2, a^2) + \Delta Q^{\pi}(\mathbf{b}^1, a^1) - \gamma \Delta Q^{\pi}(\mathbf{b}^2, a^2) \\
&\vdots \\
r^t &= Q^{\pi}(\mathbf{b}^{t-1}, a^{t-1}) - \gamma Q^{\pi}(\mathbf{b}^t, a^t) + \Delta Q^{\pi}(\mathbf{b}^{t-1}, a^{t-1}) - \gamma \Delta Q^{\pi}(\mathbf{b}^t, a^t),
\end{aligned} \tag{6.8}$$

where $r^1, \dots, r^t$ are the acquired immediate rewards. This can be written in a more compact form as

$$\mathbf{r}_t = \mathbf{H}_t \mathbf{q}_t^{\pi} + \mathbf{H}_t \boldsymbol{\Delta}\mathbf{q}_t^{\pi}, \tag{6.9a}$$

where

$$\begin{aligned}
\mathbf{r}_t &= [r^1, \dots, r^t]^{\mathsf{T}} && (6.9b) \\
\mathbf{q}_t^{\pi} &= [Q^{\pi}(\mathbf{b}^0, a^0), \dots, Q^{\pi}(\mathbf{b}^t, a^t)]^{\mathsf{T}}, && (6.9c) \\
\boldsymbol{\Delta}\mathbf{q}_t^{\pi} &= [\Delta Q^{\pi}(\mathbf{b}^0, a^0), \dots, \Delta Q^{\pi}(\mathbf{b}^t, a^t)]^{\mathsf{T}}, && (6.9d)
\end{aligned}$$

[1] Random variables are denoted with subscript, *e.g.*, a_t is a random variable at some time step t. Samples are denoted with a superscript *e.g.*, a^t is the action that was taken at time step t.

and

$$\mathbf{H}_t = \begin{bmatrix} 1 & -\gamma & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 & -\gamma \end{bmatrix}. \tag{6.9e}$$

By modelling $Q^\pi(\mathbf{b}, a)$ as a Gaussian process, $Q^\pi(\mathbf{b}, a) \sim \mathcal{GP}\left(0, k((\mathbf{b}, a), (\mathbf{b}, a))\right)$, and $\Delta Q^\pi(\mathbf{b}, a)$ as Gaussian noise, $\Delta Q^\pi(\mathbf{b}, a) \sim \mathcal{N}(0, \sigma^2)$, a similar procedure to the one outlined in Section 6.2.1 can be applied to find the posterior of $Q^\pi(\mathbf{b}, a)$, given a set of belief state-action pairs $\mathbf{B}_t$ and the observed rewards $\mathbf{r}_t$ in these belief states. The posterior is an instantiation of Eq. 6.2.

$$\begin{aligned} Q^\pi(\mathbf{b}, a)|\mathbf{r}_t, \mathbf{B}_t &\sim \mathcal{N}(\overline{Q}(\mathbf{b}, a), cov((\mathbf{b}, a), (\mathbf{b}, a))), \\ \overline{Q}(\mathbf{b}, a) &= \mathbf{k}_t(\mathbf{b}, a)^\mathsf{T}\mathbf{H}_t^\mathsf{T}(\mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\mathsf{T} + \sigma^2\mathbf{H}_t\mathbf{H}_t^\mathsf{T})^{-1}\mathbf{r}_t, \\ cov((\mathbf{b}, a), (\mathbf{b}, a)) &= k((\mathbf{b}, a), (\mathbf{b}, a)) \\ &\quad - \mathbf{k}_t(\mathbf{b}, a)^\mathsf{T}\mathbf{H}_t^\mathsf{T}(\mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\mathsf{T} + \sigma^2\mathbf{H}_t\mathbf{H}_t^\mathsf{T})^{-1}\mathbf{H}_t\mathbf{k}_t(\mathbf{b}, a) \end{aligned} \tag{6.10}$$

where $\mathbf{k}_t(\mathbf{b}, a) = [k((\mathbf{b}^0, a^0), (\mathbf{b}, a)), \dots, k((\mathbf{b}^t, a^t), (\mathbf{b}, a))]^\mathsf{T}$.

This represents the posterior at time step t of the Q-function associated with the policy π. If this was the posterior of the optimal Q-function, then for every belief state $\mathbf{b}$ the action that maximises the mean of the Q-function defines the optimal policy:

$$\pi(\mathbf{b}) = \arg\max_a \overline{Q}(\mathbf{b}, a). \tag{6.11}$$

The objective of reinforcement learning is therefore to obtain the optimal policy or to estimate the Q-function that yields the optimal policy.

In reinforcement learning algorithms there is typically a relationship between Q-function (or Value function or policy) at different time steps, which then allows the estimated version to be updated. For example, in dynamic programming algorithms such as value iteration (Algorithm 1 in Section 2.2.2) the value function in the previous step is used to re-estimate the value function at the current time step. In the case of Monte Carlo reinforcement learning approaches, the samples are acquired during the episode (a dialogue) and then the Q-function is updated at the end of the episode using the estimate from the previous episode (see Algorithm 2 in Section 2.2.2). In the temporal-difference reinforcement learning algorithm the time steps are consecutive (see the Sarsa algorithm, for example where the update occurs at every step - Algorithm 4 in Section 2.2.2).

The Gaussian process model of the Q-function described here uses a relationship between *distributions* of Q-function at two different time steps. This is in contrast to standard reinforcement learning algorithms where a relationship between *values* of the Q-function at two time steps is used.

The Q-function GP model presented has given so far a relationship between the initial time step and time step t. At the initial time step, the distribution is just a zero-mean Gaussian process with the kernel function $k((\mathbf{b}, a), (\mathbf{b}, a))$, since no data has been observed. At time step t, the distribution is the posterior distribution given the set of observed rewards $\mathbf{r}_t$ in belief state-action pairs $\mathbf{B}_t$. In the same way, the posterior can be calculated for observed rewards in the sequences of belief states of all dialogues in the training set generated with some policy. However, that would only give the posterior distribution of the Q-function associated with that policy. In order for the Q-function to be optimal, the data from which it is estimated needs to be generated in an ϵ-greedy manner, *i.e.,* either according to the current estimate of the Q-function or randomly, and the posterior needs to be re-estimated every time a new data point is observed. An example of a Gaussian process reinforcement learning algorithm that optimises the Q-function on-line in an ϵ-greedy manner is the GP-Sarsa algorithm [120]. It will be described in Section 6.2.5. Prior to that, the choice of kernel function is discussed which is followed with a discussion on reducing the computational complexity in Section 6.2.4.

6.2.3 Kernel function and hyper-parameter optimisation

The kernel function encodes prior knowledge about function correlations, and as such is crucial for the successful application of Gaussian processes to modelling a given function. Setting the kernel function manually requires a good understanding of the behaviour of the unknown function. For example, if, for any given point, it is expected that the unknown function has a similar behaviour in the vicinity of the given point, then the kernel function should account for this smoothness. In the same way, if the function is periodic, then the kernel has to capture that behaviour. Different kernel functions will be described later in this chapter for a specific task.

A problem arises when there is little prior knowledge about the domain, or when the domain is difficult to quantify in terms of kernel functions. Although under some mild conditions Gaussian processes yield the correct solution as the size of data tends to infinity [122], the rate of convergence is highly dependent on the suitability of the kernel function.

In the case of Gaussian process function approximation outlined in Section 6.2.1, the kernel function parameters may be estimated by *evidence maximisation* [37]. More specifically, if the kernel function is parameterised, the kernel parameters, also called the *hyper-parameters*, can be estimated from data in such a way that they represent the correlation that occurs in the data. This approach can be extended for the case of Gaussian process reinforcement learning in the following way.

Let $\mathbf{B}_t = [(\mathbf{b}^0, a^0), \ldots, (\mathbf{b}^t, a^t)]^\mathsf{T}$ be a sequence of belief state and action pairs for a dialogue of length t generated with policy π. Assume that the Q-function for policy π is a Gaussian process $Q^\pi(\mathbf{b}, a) \sim \mathcal{GP}\left(0, k((\mathbf{b}, a), (\mathbf{b}, a); \Theta)\right)$ with hyper-parameters Θ. The prior distribution of the Q-function in belief state-action pairs $\mathbf{B}_t$, $\mathbf{q}_t^\pi = [Q^\pi(\mathbf{b}^0, a^0), \ldots, Q^\pi(\mathbf{b}^t, a^t)]^\mathsf{T}$, is a zero-mean multivariate Gaussian distribution with covariance matrix being the Gram matrix that depends on Θ:

$$\mathbf{q}_t^\pi | \mathbf{B}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_t(\Theta)). \tag{6.12}$$

The Q-function residual in belief state-action pairs $\mathbf{B}_t$, labelled $\boldsymbol{\Delta}\mathbf{q}_t^\pi$ (Eq. 6.9d) is simply

$$\boldsymbol{\Delta}\mathbf{q}_t^\pi | \mathbf{B}_t \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}). \tag{6.13}$$

If $\mathbf{r}_t$ are the observed rewards in belief state-action pairs $\mathbf{B}_t$, from Eq. 6.9a and Eq. 6.13 the likelihood of that observation is:

$$\mathbf{r}_t | \mathbf{q}_t^\pi, \mathbf{B}_t \sim \mathcal{N}(\mathbf{H}_t \mathbf{q}_t^\pi, \sigma^2 \mathbf{H}_t \mathbf{H}_t^\mathsf{T}) \tag{6.14}$$

The marginal likelihood is obtained by integrating out $\mathbf{q}_t^\pi$:

$$p(\mathbf{r}_t | \mathbf{B}_t) = \int p(\mathbf{r}_t | \mathbf{q}_t^\pi, \mathbf{B}_t) p(\mathbf{q}_t^\pi | \mathbf{B}_t) d\mathbf{q}_t^\pi, \tag{6.15}$$

which is a convolution of two multivariate Gaussian distributions and yields a multivariate Gaussian distribution:

$$\mathbf{r}_t|\mathbf{B}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{H}_t(\mathbf{K}_t(\Theta) + \sigma^2\mathbf{I})\mathbf{H}_t^{\mathsf{T}}). \tag{6.16}$$

Using a data corpus that is labelled with belief state-action pairs $\mathbf{B}_t$[1] and rewards $\mathbf{r}_t$, the parameters $\{\Theta, \sigma\}$ can found by maximising the marginal likelihood. The solutions are however only local optima, and represent different interpretations of the correlations that occur in data.

6.2.4 Sparse approximation of Gaussian processes

Due to the matrix inversion in Eq. 6.2, the computational complexity of a Gaussian process regression is $O(t^3)$, where t is the number of data points. In the case of a dialogue system, the number of points used for estimation will be equal to the total number of turns, summed over all dialogues. This poses a serious potential computational problem, since the number can be very large. Only considering a subset of the data points and discarding the rest is not sufficient for the Q-function to be adequately estimated even for a discrete space MDP. This is exacerbated if the space is continuous (like here), as the same state is never visited twice. It is therefore important to take into account the contribution each part of the belief state space brings to the final estimation. For this reason, a sparse approximation of Gaussian processes is needed. This should, on one hand, ensure that all the data points are taken into account, and, on the other hand, reduce the computational complexity. This section explains the Fully Independent Training Conditional methods and the kernel span sparsification method.

The cubic increase in complexity with the number of visited belief states is the Achilles' heel of Gaussian process regression. Significant research effort has been invested into solving this problem, and a number of methods have been developed [123]. One particularly good approach is the Fully Independent Training Conditional (FITC) algorithm[2] [124]. The main idea behind this approach is to assume that the function values at the data points are conditionally independent, given some set of *support points*. The likelihood can then be approximated with a Gaussian distribution that has a diagonal covariance matrix. This reduces the computational effort required

[1]Note that this is obtained automatically in the process of belief update.

[2]Originally named Sparse Gaussian Process using Pseudo-inputs (SGPP).

to calculate the covariance matrix of the posterior from $O(t^3)$ to $O(tm^2)$, where m is the number of support points. If the number of support points is significantly smaller than the number of data points, this approach is very effective for reducing the computational cost. The main problem of this approach, however, is to find a suitable way of selecting the support points.

The set of support points is typically chosen using some heuristics to select a small subset of the data points. It is also possible to select the subset using a probabilistic approach, *e.g.,* by maximising the marginal likelihood. However, both of these methods assume that a large set of training points is available. This restricts them to cases where the data points are available from the outset, as is the case in off-line methods, but not normally in on-line methods.

An alternative algorithm which approximates the Gaussian process without first obtaining a set of support points is the kernel span sparsification method described in [125]. As it visits the belief state space, it keeps track of only a small subset of the visited points, which are used to approximate the kernel function in such a way that the overall computational complexity can be reduced. Chapter 8 will give some suggestions for how this algorithm can be extended to provide a set of support points so that the FITC algorithm can be used online.

A kernel function can be thought of as a dot product of a (potentially infinite) set of feature functions $k((\mathbf{b}, a), (\mathbf{b}, a)) = \langle \boldsymbol{\phi}(\mathbf{b}, a), \boldsymbol{\phi}(\mathbf{b}, a) \rangle$ where $\boldsymbol{\phi}(\mathbf{b}, a)$ is the vector of feature functions $[\phi_1(\mathbf{b}, a), \phi_2(\mathbf{b}, a), \ldots]^\mathsf{T}$. Any linear combination of feature vectors $\{\boldsymbol{\phi}(\mathbf{b}^0, a^0), \ldots, \boldsymbol{\phi}(\mathbf{b}^t, a^t)\}$ for a given set of points $\{(\mathbf{b}^0, a^0), \ldots, (\mathbf{b}^t, a^t)\}$ is called the *kernel span.* The aim is to find the subset of points that approximates the kernel span. These points are called the *representative points* and the set of representative points is called the *dictionary* $\mathcal{D} = \{(\tilde{\mathbf{b}}^0, \tilde{a}^0), \ldots, (\tilde{\mathbf{b}}^m, \tilde{a}^m)\}$.

The sparsification parameter ν places a threshold on the squared distance between the feature function span at the representative points, and the true feature function value at each visited point. Every time the threshold is exceeded, a new point is added to the dictionary. If $(\mathbf{b}^t, a^t)$ is the current data point then:

$$\min_{\mathbf{g}_t} \parallel \sum_{j=0}^{m} g_{tj} \boldsymbol{\phi}(\tilde{\mathbf{b}}^j, \tilde{a}^j) - \boldsymbol{\phi}(\mathbf{b}^t, a^t) \parallel^2 \leq \nu, \tag{6.17}$$

where $\mathbf{g}_t = [g_{t1}, \ldots, g_{tj}]$ is a vector of coefficients and m is the size of the current dictionary, $\mathcal{D} = \{(\tilde{\mathbf{b}}^0, \tilde{a}^0), \ldots, (\tilde{\mathbf{b}}^m, \tilde{a}^m)\}$. It can be shown [125] that this is equivalent to:

$$\min_{\mathbf{g}_t} \left(k((\mathbf{b}^t, a^t), (\mathbf{b}^t, a^t)) - \tilde{\mathbf{k}}_{t-1}(\mathbf{b}^t, a^t)^\intercal \mathbf{g}_t \right) \leq \nu, \tag{6.18}$$

where $\tilde{\mathbf{k}}_{t-1}(\mathbf{b}^t, a^t) = [k((\mathbf{b}^t, a^t), (\tilde{\mathbf{b}}^0, \tilde{a}^0)), \ldots, k((\mathbf{b}^t, a^t), (\tilde{\mathbf{b}}^m, \tilde{a}^m))]^\intercal$. It can also be shown that the expression on the left side of Eq. 6.18 is minimised when $\mathbf{g}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\mathbf{b}^t, a^t)$, where $\tilde{\mathbf{K}}_{t-1}$ is the Gram matrix of the current set of representative points. If the threshold ν is exceeded then $(\mathbf{b}^t, a^t)$ is added to the dictionary, otherwise the dictionary stays the same. This constitutes the sparsification criteria and allows for approximation of the posterior in complexity in the following way.

Since the kernel function is the dot product of the feature functions, the Gram matrix is $\mathbf{K}_t = \mathbf{\Phi}_t^\intercal \mathbf{\Phi}_t$ where $\mathbf{\Phi}_t = [\boldsymbol{\phi}(\mathbf{b}^0, a^0), \ldots, \boldsymbol{\phi}(\mathbf{b}^t, a^t)]$. Then, the feature function values for each point are approximated as the linear combination of the representative points $\boldsymbol{\phi}(\mathbf{b}^i, a^i) \approx \sum_{j=1}^m g_{ij} \boldsymbol{\phi}(\tilde{\mathbf{b}}^j, \tilde{a}^j)$, for all $i \in 0, \ldots, t$. Also, the Gram matrix is approximated as $\mathbf{K}_t = \mathbf{\Phi}_t^\intercal \mathbf{\Phi}_t \approx \mathbf{G}_t \tilde{\mathbf{K}}_t \mathbf{G}_t^\intercal$, where $\mathbf{G}_t = [\mathbf{g}_1, \ldots, \mathbf{g}_t]$. In a similar manner, $\mathbf{k}_t(\mathbf{b}, a) \approx \mathbf{G}_t \tilde{\mathbf{k}}_t(\mathbf{b}, a)$. This allows for the posterior (Eq. 6.10)to be approximated:

$$\begin{aligned} Q(\mathbf{b}, a) | \mathbf{B}_t, \mathbf{r}_t \sim \mathcal{N}\left(\tilde{\overline{Q}}(\mathbf{b}, a), \widetilde{cov}((\mathbf{b}, a), (\mathbf{b}, a))\right), \\ \tilde{\overline{Q}}(\mathbf{b}, a) = \tilde{\mathbf{k}}_t(\mathbf{b}, a)^\intercal (\tilde{\mathbf{H}}_t^\intercal (\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\intercal + \sigma^2 \tilde{\mathbf{H}}_t \tilde{\mathbf{H}}_t^\intercal)^{-1} \mathbf{r}_t), \\ \widetilde{cov}((\mathbf{b}, a), (\mathbf{b}, a)) = k((\mathbf{b}, a), (\mathbf{b}, a)) \\ - \tilde{\mathbf{k}}_t(\mathbf{b}, a)^\intercal (\tilde{\mathbf{H}}_t^\intercal (\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\intercal + \sigma^2 \tilde{\mathbf{H}}_t \tilde{\mathbf{H}}_t^\intercal)^{-1} \tilde{\mathbf{H}}_t) \tilde{\mathbf{k}}_t(\mathbf{b}, a), \end{aligned} \tag{6.19}$$

where $\tilde{\mathbf{H}}_t = \mathbf{H}_t \mathbf{G}_t$. It can be shown that this reduces the complexity to $O(tm^2)$, where m is the number of representative points.

This sparsification method can be effectively utilised in an on-line algorithm where the data is obtained sequentially. As can be seen from Algorithm 8, the process starts with just one data point which constitutes the initial dictionary. Every time a new data point is encountered, a vector of kernels between the current dictionary points and the new point is calculated. The linear combination of the vectors of the feature functions at the current dictionary points that is the closest to the vector of the feature functions at the new data point is then computed. If the sparsification threshold is exceeded, the new data point is added to the dictionary with all coefficients set to zero, except for the last one which corresponds to the newly added dictionary element.

Algorithm 8 On-line kernel span sparsification algorithm

```
1: Initialise $\mathcal{D} \leftarrow \{(\mathbf{b}, a)\}$, $m \leftarrow 1$, $\tilde{\mathbf{K}}^{-1} \leftarrow [1/k((\mathbf{b}, a), (\mathbf{b}, a))]$, $\mathbf{G} \leftarrow [1]$,
   $t \leftarrow 0$
2: repeat
3:    $t \leftarrow t + 1$
4:    Get new belief state-action pair $(\mathbf{b}', a')$
5:    Compute $\tilde{\mathbf{k}}(\mathbf{b}', a')$
6:    $\mathbf{g} \leftarrow \tilde{\mathbf{K}}^{-1}\tilde{\mathbf{k}}(\mathbf{b}', a')$
7:    $\delta \leftarrow k((\mathbf{b}', a'), (\mathbf{b}', a')) - \tilde{\mathbf{k}}(\mathbf{b}', a')^{\mathsf{T}}\mathbf{g}$
8:    if $\delta \geq \nu$ then
9:       $\mathcal{D} \leftarrow \{(\mathbf{b}', a')\} \cup \mathcal{D}$
10:      Recompute $\tilde{\mathbf{K}}^{-1}$
11:      $m \leftarrow m + 1$
12:      $\mathbf{G} \leftarrow \begin{bmatrix} \mathbf{G} & \mathbf{z} \\ \mathbf{z}^{\mathsf{T}} & 1 \end{bmatrix}$
13:   else
14:      $\mathbf{G} \leftarrow \begin{bmatrix} \mathbf{G} \\ \mathbf{g}^{\mathsf{T}} \end{bmatrix}$
15:   end if
16: until $t == T$
```

This sparsification method turns a non-parametric method into a parametric method. More precisely, the kernel is now approximated using only a limited number of points, which is equivalent to defining a functional basis for the kernel function, and defining the kernel to be a finite linear combination of that basis which can be shown to be equivalent to parameterising the Q-function [120]. This may limit the accuracy of the solution since the method finds the optimal solution only within the span of the kernel basis functions. However, the fact that the basis is chosen dynamically still allows for more appropriate functions than when using a fixed basis.

One advantage of the approach is that it enables non-positive definite kernel functions to be used in the approximation. This is due to the fact that the sparsification method essentially changes the kernel function in a manner which ensures that the approximated Gram matrix is positive definite. According to [126], this is sufficient to guarantee that the model remains well-defined. What is more, using a sparse approximation sometimes yields better results than the full Gaussian process model since the approximated kernel function might suit the task better than the original kernel function [127].

When using this approximation technique, even though the algorithm approximates a Gaussian process model, the result may not be a Gaussian process model itself. This poses a potential issue since there is no guarantee that the covariance in the Eq. 6.19 is positive. However, this can be easily amended by replacing $k((\mathbf{b},a),(\mathbf{b},a))$ in the Eq. 6.19 with $\tilde{\mathbf{k}}_t(\mathbf{b},a)^\intercal \tilde{\mathbf{K}}_t^{-1}\tilde{\mathbf{k}}_t(\mathbf{b},a)$ and the covariance of the posterior becomes:

$$\widetilde{cov}((\mathbf{b},a),(\mathbf{b},a)) = \tilde{\mathbf{k}}_t(\mathbf{b},a)^\intercal(\tilde{\mathbf{K}}_t^{-1} - \tilde{\mathbf{H}}_t^\intercal(\tilde{\mathbf{H}}_t\tilde{\mathbf{K}}_t\tilde{\mathbf{H}}_t^\intercal + \sigma^2\tilde{\mathbf{H}}_t\tilde{\mathbf{H}}_t^\intercal)^{-1}\tilde{\mathbf{H}}_t)\tilde{\mathbf{k}}_t(\mathbf{b},a). \tag{6.20}$$

In this way, the kernel function is not only approximated at the places which are necessary to keep the computational cost down (such as the computing inverse of the Gram matrix), but it is replaced with its approximation wherever it appears in Eq. 6.10.

Thus, the original Gaussian process is approximated with another Gaussian process where the kernel function is in a form which allows for the inverse of the Gram matrix to be calculated in a more efficient way. This ensures that the GP model for the Q-function remains well-defined.

The only downside is that sometimes these approximations can lead to an overly confident estimate [123]. In order to alleviate this, in the set-up that will be used here the estimate of the variance from Eq. 6.19 will be used as long as it gives a positive value, otherwise Eq. 6.20 will be used.

6.2.5 GP-Sarsa

The previous section has shown how the Q-function Q^π for some policy π can be modelled as a Gaussian process (see Eq. 6.19). This requires that a kernel over both belief state and action spaces is defined. If the action space $\mathcal{A}$ is discrete, the kernel function can be composed of a kernel over belief states and a kernel over actions, $k_{\mathcal{B}\times\mathcal{A}}((\mathbf{b},a),(\mathbf{b}',a')) = k_{\mathcal{B}}(\mathbf{b},\mathbf{b}')k_{\mathcal{A}}(a,a')$. This allows for $k_{\mathcal{B}}(\mathbf{b},\mathbf{b}')$ to be chosen from a set of standard kernel functions for continuous spaces and $k_{\mathcal{A}}(a,a')$ from a set of standard kernel functions for discrete spaces, which makes the task of selecting a kernel easier[1].

[1]Refer to [37] for a list of standard kernel functions for different input spaces.

The GP-Sarsa algorithm estimates the optimal Q-function by modelling it as a Gaussian process and updating the posterior after each quintuple $(b(s_t), a_t, r_t, b(s_{t+1}), a_{t+1})$ is observed, where actions are taken ϵ-greedily [120]. This relates to the original Sarsa for step size parameter $\lambda = 1$ (see Algorithm 4 in Section 2.2.1). In the Bayesian framework, instead of using a step size parameter, the underlying distribution controls how much each reward observation should contribute to the function estimate, based on how likely the particular reward observation is.

The sequential nature of on-line learning allows for a more efficient calculation of the matrix inversion in Eq. 6.19. A full description of this is given in Algorithm 9 in Appendix D.1.

6.2.6 Active learning in Gaussian process reinforcement learning

Active learning is a machine learning method for efficient data selection [128, 129]. The main idea behind active learning is to select only the data points that contribute the most to the estimate. In this way it can reduce the cost of data labelling or speed up the learning process, depending on the application. Instead of random data selection, active learning selects data points according to some *utility function*. Although it is capable of accelerating the learning process, active learning suffers from an intrinsic problem called the *sampling bias* [130]. More specifically, when data points are selected at random they accurately represent the distribution. However, once they are chosen according to some utility function they are biased towards the heuristics that the function encodes, which may lead to inconsistencies in the estimation.

In on-line reinforcement learning one of the main issues is the exploration-exploitation trade-off. ϵ-greedy learning normally requires random exploration with probability ϵ (Section 2.2.1). However, such random exploration can be inefficient as not all parts of the space are equally informative when it comes to the Q-function approximation. Therefore, adopting an active learning approach to exploration could yield a significant improvement in the speed of learning. The utility function normally includes a measure of the information gain [128], through which various heuristics can be incorporated. What is particularly appealing about Gaussian process reinforcement learning is that it provides a measure of uncertainty at each point of the space. This can then be used directly in the active learning utility function [118]. This enables the model to explore the parts of the space it is less certain about.

Active learning was previously used in POMDP dialogue management as a way of selecting *meta-queries* efficiently [131]. Meta-queries are a special type of actions that the system can take in order to ask the user directly whether taking a particular action is good. This may be important in circumstances of high uncertainty and when taking a different action may be risky. In order to minimise the risk and at the same time minimise the number of times the meta-queries were asked, active learning was applied. On average, it showed better performance than alternative methods.

Here, active learning was used for more efficient exploration. During exploration, actions are chosen based on the variance of the GP estimate for the Q-function and during exploitation, actions are chosen based on the mean:

$$a = \begin{cases} \arg\max_a \overline{Q}(\mathbf{b}, a) & \text{with probability } 1-\epsilon\text{, exploitation} \\ \arg\max_a \mathit{cov}\left((\mathbf{b}, a), (\mathbf{b}, a)\right) & \text{with probability } \epsilon\text{, exploration} \end{cases} \tag{6.21}$$

6.3 Gaussian process RL on a toy problem

In order to demonstrate how this methodology can be applied to a dialogue system, the idea is first illustrated by a simple problem, the VoiceMail dialogue task [70].

6.3.1 VoiceMail dialogue task

The state space of the VoiceMail task consists of three states: the user asked for the message to either be saved or deleted, or the dialogue ended. The system can take three actions: ask the user what to do, save the message or delete the message. Transitions to the final state are deterministic depending on whether the user ended the call or the system saved or deleted a message. However, the observation of what the user wants is corrupted with noise, and the system therefore receives an input that does not always correlate exactly with what the user intended. For that reason, under the POMDP framework, the probability of each state is estimated in every dialogue turn.

In both learning and evaluation, a simulated user is used which makes errors with probability 0.3, and terminates the dialogue after at most 10 turns. It assigns a positive reward of 10 in the final state if the system has performed a correct action, otherwise it gets a penalty of -100. Each intermediate state receives a reward of -1. In order to keep the problem simple, a model defining transition and observation probabilities is assumed so that the belief can be updated easily. The actual policy optimisation, however is performed in an on-line fashion.

6.3.2 Choosing a kernel for the VoiceMail task

Choosing a kernel function for a particular task requires knowledge about the correlations in the Q-function. Not every function of two belief states is a valid kernel function so choosing the right kernel function is a difficult task in itself. The intention here, however, was not to examine which kernel is best suited for a particular dialogue task, but instead to examine the potential of GP-Sarsa to speed up the learning process when compared to the Monte Carlo control algorithm, a standard non-parametric algorithm. Therefore, only a small number of kernel functions were compared. For the VoiceMail task this includes two standard kernel functions and a kernel function that was specially constructed for this task.

The polynomial kernel function is defined as:

$$k(\mathbf{b}, \mathbf{b}'; \sigma_0, p) = (\langle \mathbf{b}, \mathbf{b}' \rangle + \sigma_0^2)^p, \tag{6.22}$$

where $\langle \cdot, \cdot \rangle$ is the dot-product, with hyper-parameters $\sigma_0 >= 0$ and $p > 0$. If σ_0 is zero this is the homogeneous polynomial kernel and in the case $p = 1$ and $\sigma_0 = 0$ this is the linear kernel.

A parametrisation of the linear kernel that places varying relevance on different elements of the belief state is:

$$k(\mathbf{b}, \mathbf{b}'; \rho_1, \ldots, \rho_{|\mathbb{S}|}) = \sum_{i=1}^{|\mathbb{S}|} \frac{b_i b_i'}{\rho_i^2} \tag{6.23}$$

where hyper-parameter ρ_i determines the relevance of the ith element of the belief state vector $\mathbf{b}$; the larger this value is the less relevant that element is. If these hyper-parameters are optimised, this kernel is called linear with *automatic relevance determination* [132].

The Gaussian kernel function[1] is defined as:

$$k(\mathbf{b}, \mathbf{b}'; p, \sigma_k) = p^2 \exp\left(-\frac{\|\mathbf{b}-\mathbf{b}'\|^2}{2\sigma_k^2}\right), \tag{6.24}$$

where σ_k determines how close the points have to be for the values of the function to be correlated, and p defines the prior variance at each data point since $k(\mathbf{b}, \mathbf{b}) = p^2$. The main advantage of the Gaussian kernel over the polynomial kernel is that it is a dot product of an infinite vector of features functions [37] which gives it the potential to model covariances better. On the other hand, while the polynomial kernel is non-stationary, the Gaussian kernel is a stationary kernel. This means that that the covariances only depend on the distance between two belief states rather than on the part of the space where the two belief states are.

The algorithm for hyper-parameter optimisation of Gaussian process regression in [133] was adapted to obtain the optimal hyper-parameters for parametrised kernel functions of the GP model for Q-function using a dialogue corpus labelled with belief states, actions and rewards (Section 6.2.3).

The third kernel considered here is the scaled norm kernel defined as:

$$k(\mathbf{b}, \mathbf{b}') = 1 - \frac{\|\mathbf{b}-\mathbf{b}'\|^2}{\|\mathbf{b}\|^2\|\mathbf{b}'\|^2}. \tag{6.25}$$

This function is not positive definite and as such is not a valid kernel. However, it can be used in combination with the kernel span sparsification method, as discussed in Section 6.2.4. This function defines positive correlations at the points that are close to each other, and negative correlations otherwise. This is particularly useful for the VoiceMail task, where, if two belief states are very different, taking the same action in these belief states generates a negatively correlated return. For example, assume that one knows the value of the Q-function corresponding to the delete action when there is a high probability that the user wants it to be deleted. It is then reasonable to assume that taking the delete action when there is a high probability that the user wants the message to be saved will be negatively correlated with that (known) Q-value. Thus, during the process of learning, experiencing the case where the system deletes when there is a high proba-

[1] Also called the squared exponential kernel function in the literature.

bility that the user wants the message to be deleted also gives an estimate of the Q-function and an associated uncertainty for the case when there is low probability that the user wants the message to be deleted and the system takes the delete action, without actually experiencing that case.

The kernel function can also be defined over discrete variables. A simple example of such a kernel is the δ kernel [134], which is used here to define a kernel over actions:

$$k(a, a') = 1 - \delta_a(a'), \tag{6.26}$$

where δ_a is the Kronecker delta function.

6.3.3 Training set-up and evaluation on VoiceMail

In order to assess the performance of GP-Sarsa, it was compared to the standard grid-based Monte Carlo control algorithm[1] (Algorithm 2, Section 2.2.1). The policies were trained in interaction with a simulated user.

The intention was not only to test which algorithm for optimising the Q-function on-line yields the best policy performance, but also to examine the rate of convergence to the optimal policy. Since the underlying transition and observation probabilities are known and the action space, the state space and the observation space are small, it is possible to find the optimal policy in an off-line fashion without any approximations. The optimal policy was obtained using the POMDP solver toolkit [135]. This toolkit implements the Value iteration algorithm to solve the POMDP off-line (as outlined in Section 2.2.5). Then, that policy was used to generate 300 dialogues in interaction with the simulated user. These dialogues served as data to optimise the kernel hyper-parameters.

All the algorithms use an ϵ-greedy approach where the exploration rate ϵ was fixed at 0.1. The learning process depends greatly on the actions that are taken during exploration. If early on during the training process due to a fortunate choice of actions, the system discovers a path that generates high rewards, then convergence is reached sooner. Therefore, the learning process itself is a random process. In order to mitigate against this, the following procedure is adopted. For every training set-up, exactly the same number of training iterations were performed using 1000 different random generator

[1] A grid-based Sarsa algorithm (Algorithm 4, Section 2.2.1) was also examined within two dialogue domains, but it exhibited inconsistency in convergence and sensitivity to the choice of the step-size parameter.

seedings. After every 20 dialogues the resulting 1000 partially optimised policies were evaluated. Each of them was tested on 1000 dialogues. The average reward of these 1000 dialogues provides just one point in Fig. 6.1, which shows the learning progress.

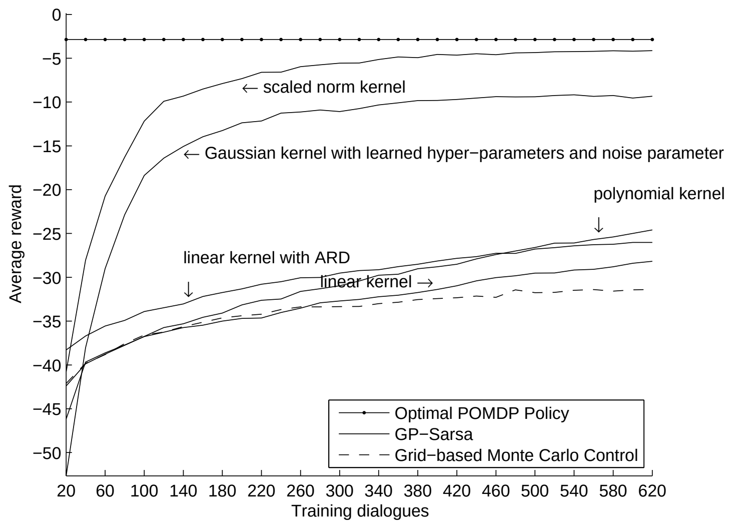


Figure 6.1: Evaluation results on the VoiceMail task.

The grid-based Monte Carlo control algorithm used a Euclidean distance to generate the grid by adding every point that was further than 0.01 from other points since that one may be considered representative of a new region. As can be seen from Fig 6.1, the grid-based Monte Carlo control algorithm has a relatively slow convergence rate.

For GP-Sarsa the noise variance of the residual σ (see Section 6.2.2) was hand-tuned at 1.0, except in the case of the Gaussian kernel, where it was optimised along with the other parameters. The sparsification threshold ν (Section 6.2.4) was set to 0.01. From Fig. 6.1, the linear kernel (Eq. 6.22) exhibited a learning rate similar to the Monte Carlo control algorithm in the first 300 training dialogues, but thereafter learnt somewhat faster. A non-homogeneous second order polynomial was also examined ($\sigma_0 = 1$ and $p = 2$). However its ability to model covariances is only marginally better than the linear kernel. The linear kernel with automatic relevance determination (ARD) from Eq. 6.23, performs slightly better in the early stages of the learning process. The Gaussian kernel, however, with optimised hyper-parameters ($p = 5.48$, $\sigma_k = 0.5$) and a learnt parameter for the noise residual ($\sigma = 14.12$) achieves a much faster learning rate. The best performance was obtained using the kernel from Eq. 6.25. Not only did it achieve close to optimal performance in 400 dialogues, but its convergence rate was higher in comparison with the other methods.

The Q-value functions found by GP-Sarsa with the scaled norm kernel and the POMDP solver are presented in Fig. 6.2 and Fig. 6.3, respectively. GP-Sarsa did not estimate the optimal Q-value function very well and the error bars suggest an overly confident approximation. This might be a result of the sparsification process, since sparse approximation methods can lead to overly confident estimates [123]. However, it was able to yield better approximations of the optimal Q-values at the points where the Q-value function associated with one action intersects with the Q-value function associated with another action. This corresponds to one action having higher expected return than the other in that part of the belief space, which is precisely what is important in policy optimisation.

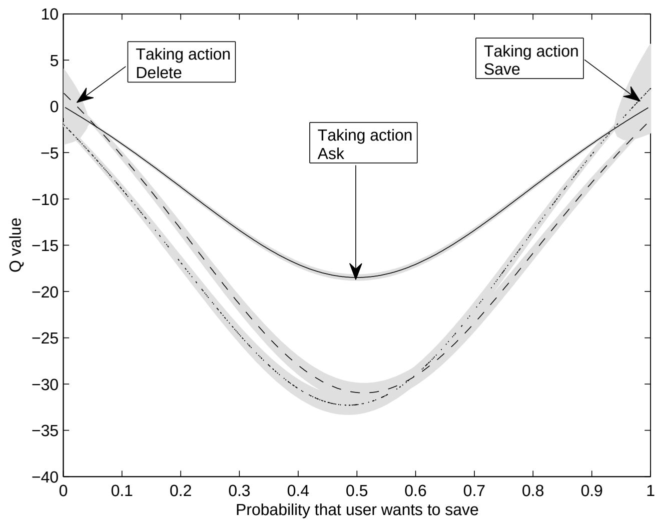


Figure 6.2: Approximated Q-value function using GP-Sarsa with scaled norm kernel. The grey area indicates a 95% confidence interval.

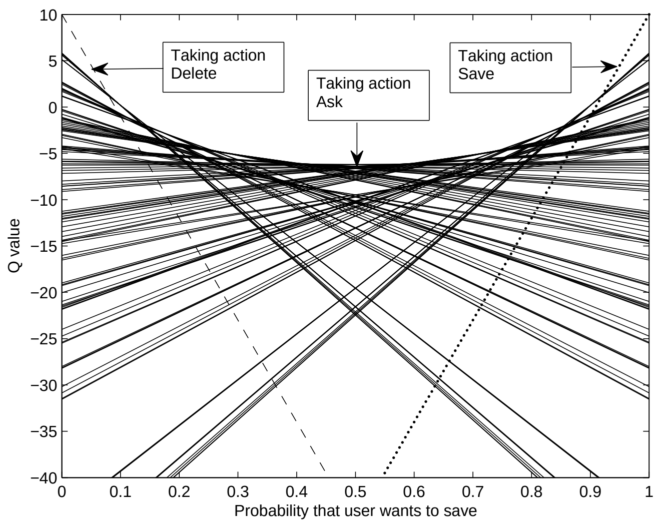


Figure 6.3: Upper envelope of optimal Q-value function using POMDP solver.

6.4 Gaussian process RL on a real-world task in the HIS dialogue manager

The results on the toy problem presented above suggest that applying Gaussian processes to reinforcement learning can speed up policy optimisation. In order to verify this result on a real-world task, the Hidden Information State dialogue manager was extended to incorporate GP-Sarsa, and the experiments were carried out in the CamInfo domain (Section C.1). For each experiment, the policy was trained and evaluated in interaction with the agenda-based simulated user as described in Section 3.4.

6.4.1 Choosing a kernel for the CamInfo task

Performing learning on the belief state space removes the need for defining a summary space as well as the heuristics for mapping back from summary to master space, hence enabling the dialogue manager to learn the full scope of behaviour. However, it requires a kernel function to be defined over belief state space, which, in the case of the HIS system, requires a kernel that can be defined over trees. An example of such a kernel is given in [136]. However, directly applying such a kernel is not possible since the partitions are trees of variable depth. In addition, the size of the belief state vector varies so that a variant of rational kernels [137] may be required. These techniques are outside the scope of this thesis and will be left for future work. For the purpose here, which is mainly to show contrasts between different algorithms, the summary space is sufficiently complex.

In contrast to the summary space defined in Section 3.3.5, the summary state used here is four dimensional. This is one dimension less than the space used there because intermediate experimentation showed that the hypothesis status (defined in Table 3.5 in Section 3.3.3) did not contribute significantly to the policy optimisation process. The hypothesis status is a compressed form of the grounding information associated with a given hypothesis. Since the mapping from summary to master space (Section 3.3.5) already makes use of the grounding information when constructing the system's dialogue act from the summary action, the hypothesis status is redundant in the summary space. Note that the summary space used here consists of two elements that are continuous (the probability of the top two hypotheses) and two discrete elements – the partition status, which relates to the portion of the database entries that matches the top partition, and the last user action type associated with the top hypothesis. The summary action space is discrete, consisting of eleven elements as defined in Table 3.6 in Section 3.3.5. While it would be desirable to extend both the summary state and the summary action state space this would require a drastic change in the mapping from the summary to the master space. This mapping is based on heuristics and its implementation design goes beyond topic of this thesis.

In order to apply the GP-Sarsa algorithm, a kernel function needs to be specified for both the summary state space and the summary action space (see Section 6.2.5). The nature of this space is quite different from the one in the toy problem, with the main difference being the use of the master to summary space mapping. Intermediate experimentation showed that applying a kernel that has negative correlations, such as the one in Eq. 6.25, does not yield good results. The reason for this may lie in the summary to master space mapping. For a given summary action, the mapping procedure finds the most appropriate action to perform if such an action exists. This can lead to a lower reward if the summary action is not adequate, but would rarely lead to negatively correlated rewards. In this way, the heuristics in the summary to master space mapping alleviate the effect of taking inadequate summary actions. Furthermore, hand-tuning the parameters of the Gaussian kernel (Eq. 6.24) against average reward did not produce good results. Polynomial kernels (Eq. 6.22) assume that the elements of the space are features. Due to the manner in which the probability is maintained over this very large state-space, the continuous variables may potentially encode more information than in the simple toy problem. For example, different partitioning of the user goal leads to different probabilities of the top hypothesis which may be useful information for action selection. Both linear and second order polynomial kernels were therefore examined. For discrete features of the summary space, the δ-kernel was used (Eq. 6.26).

The sparsification threshold ν was hand-tuned to 0.35 to ensure that the number of elements in the dictionary remained low. The noise parameter σ was fixed at 2.5. As defined in Section 6.2.2, σ is the variance of the residual, ΔQ. From Eq. 6.6 it can be seen that σ directly relates to how much variability is expected in the return. In contrast to the toy problem, the optimisation process here operates in a summary space, so the return naturally exhibits more variability. For example, if one return is associated with one belief point and another return with another belief point and if these belief points are mapped to the same summary point, the return associated with that summary point will vary. The return also varies due to the complexity of the problem, especially in the beginning of the learning process.

6.4.2 Training set-up and evaluation on CamInfo

In training and testing, the agenda-based user simulator had the same parameter set-up as defined in Section 3.4.2. The reward function gave a reward of 20 for successful dialogues, the exploration factor was 0.1 and the discount factor was 0.95. Only training on zero error rate was considered, since it is easier to observe the policy performance improvement during optimisation if there are no changes in the training conditions and no noise is introduced.

The extended grid-based Monte Carlo control algorithm (Algorithm 7 in Section 5.4) provided the baseline method for testing the speed of learning. The Euclidian distance metric was used to generate the grid (Section 3.3.5) and the threshold was set to 0.35. An intermediate examination showed that using a threshold of 0.01 as in Section 3.4.3 yields a slow convergence rate, so the threshold was increased to provide a better baseline. In addition, the GP-Sarsa algorithm was augmented to make use of the N-best back-off strategy of dealing with the invalid summary actions (Section 5.4) in order to establish a fair comparison.

To measure the rate at which each algorithm learns, while at the same time minimising the effect of randomness during learning, a similar evaluation set-up to the one presented in Section 6.3.3 was adopted. The training was performed 1000 times, each with a different random seeding. After every 200 dialogues each of the 1000 partially optimised policies was evaluated on 1000 dialogues. The averaged reward and success rates are shown in Figs. 6.4 and 6.5 respectively.

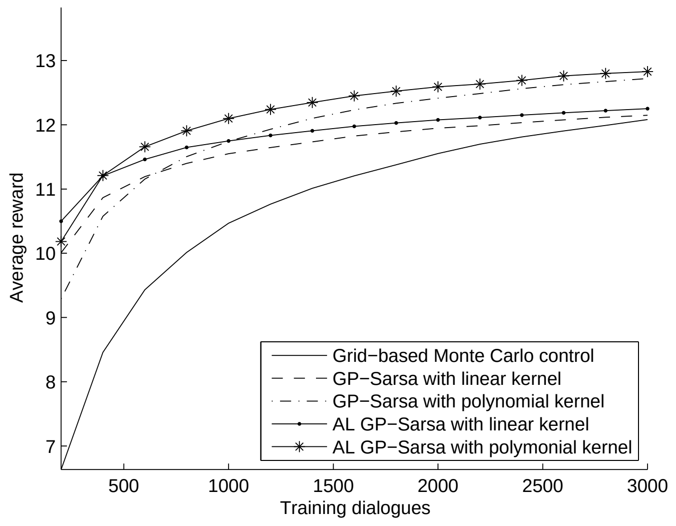


Figure 6.4: Evaluation results on CamInfo task – reward.

The results show that during the first 3000 dialogues, the GP-based method learns much faster than the Monte Carlo control algorithm. While the second-order polynomial kernel performed worse than the linear kernel at the early stages of learning, it obtained a higher average reward after 1000 dialogues. The second-order polynomial kernel results in a more complex Gaussian Process model, so it may only model correlations well once the policy reaches a certain performance level. Previous research has shown that changing kernel parameters on-line during learning yields better performance [138]. In the same way, different kernel functions may be better

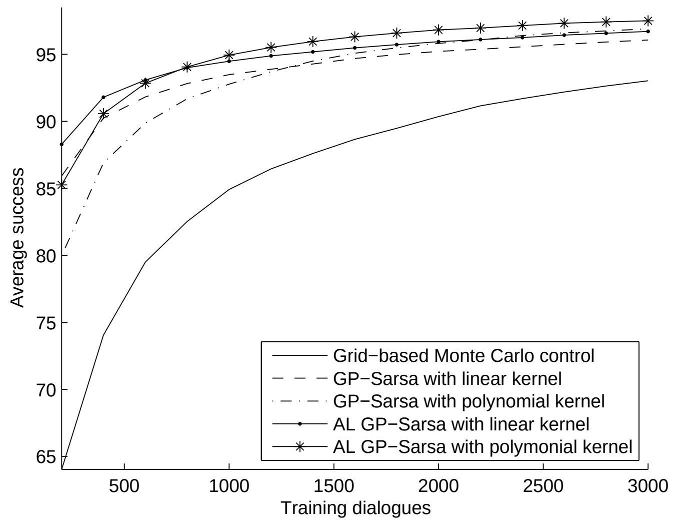


Figure 6.5: Evaluation results on CamInfo task – success.

suited for different stages of learning.

Also, active learning was applied to both GP-Sarsa with a linear kernel as well as with a second-order polynomial kernel. There, the actions are selected during exploration based on the estimated uncertainty (see Eq. 6.21 in Section 6.2.6). In both cases, the results show accelerated learning during the initial training phase. After performing many iterations in an incremental noise configuration (Section 3.4.3) both the GP-Sarsa and the grid-based Monte Carlo algorithms converge to the same performance.

Not only did GP-Sarsa have a faster learning rate at the beginning of training, but the success rates and the rewards that it generates suggest that it could potentially be used as a method for on-line learning in direct interaction with real users. There are two main reasons why standard RL algorithms are not normally used to train the policy in direct interaction with real people. First, they need a large number of dialogues and second, the policy performance in the early stages of learning is so poor that it is not considered acceptable for interaction with real people. Here however, the success rate surpasses 85% after only 200 dialogues, which suggests that GP-Sarsa is able to overcome these problems.

6.5 Summary

This chapter has described how Gaussian processes in reinforcement learning can be applied successfully to dialogue management. The main motivations for using Gaussian processes for reinforcement learning in a POMDP dialogue manager are to speed up the learning process and to model the uncertainty of the estimates. Various aspects of Gaussian process reinforcement learning have been discussed, such as the choice of kernel function, sparse approximations of Gaussian processes, and the use of active learning for efficient exploration of the summary space. GP-Sarsa was implemented on a toy dialogue problem and the use of different kernel functions was investigated, showing that fast convergence can be achieved. It was also demonstrated how kernel hyper-parameters, used for on-line policy optimisation, can be learnt from a dialogue corpus, thus creating a bridge between supervised and reinforcement learning methods in dialogue management. GP-Sarsa was also applied to a real-world dialogue task showing that, on average, this method is able to learn much faster than a grid-based algorithm and that an active learning setting can further accelerate policy optimisation reaching more than 85% success rate after only 200 dialogues.

This does not only show that the GP-Sarsa is able to learn faster, but also that it also allows more aspects of the dialogue management behaviour to be learned. This means that given an suitable kernel the learning can be performed directly on the full belief space. However, this remains to be investigated.

Another important aspect of kernel selection is the optimisation of the kernel hyper-parameters. Estimating them from a dialogue corpus for a real-task is a part of future work.

The Gaussian process model does not only provide a probabilistic framework for fast policy optimisation but can also be used for policy adaptation to different users. The next chapter will examine how the uncertainty measure that the Gaussian process is estimating can be utilised for this purpose.

Chapter 7

Adaptation to different user types

7.1 Introduction

In this chapter, strategies for adaptation to different user types are studied in the context of Gaussian process reinforcement learning for dialogue management as introduced in the previous chapter, where the Q-function is modelled as a Gaussian process. The chapter starts with a discussion of the different approaches to adaptation in dialogue management. A short-term adaptation strategy that uses uncertainty estimates from a Gaussian process is presented in Section 7.3. In Section 7.4, a description of the user simulator that is able to simulate different user types is given. Then, in Section 7.5 evaluation results on the Hidden Information State system are presented. Section 7.6 investigates the differences in performances between a policy trained on several users and tested on a particular user and the adaptation to that particular user. A summary of the chapter with the main conclusions is given in Section 7.7.

7.2 Approaches to adaptation in dialogue management

In dialogue management, adaptation can be viewed as a process of improving the action selection on a different condition to the one the policy was originally trained on. For example, if the policy is trained on a simulated user and then adapted to a real user. Another example is a policy that is trained for an expert user and is then adapted to a naïve user. Adaptation is *supervised* if either the true user actions or the rewards are known. Otherwise, the adaptation is *unsupervised*.

While adaptation has been extensively studied in speech recognition (see an overview in [28]), in spoken dialogue systems it is still relatively novel and covers a wide range of possible research topics [59, 139, 140, 141]. Broadly, adaptation approaches in spoken dialogue systems can be divided into *long-term* adaptation, when adaptation takes place only after a number of dialogues, and *short-term* adaptation, where adaptation takes place within a single dialogue.

Reinforcement learning in principle provides a mechanism for long-term, supervised adaptation. For example, if the estimate of the reward during human-computer interaction is available, the system can re-estimate the Q-function, over time leading to a better dialogue policy. In this way, the policy trained with the simulated user can be further improved in direct interaction with real users. Long-term adaptation is also useful to adapt to a single user in such applications where it is expected that the same user will have multiple interactions with the system. Long-term adaptation, however, requires a reward to be associated with every dialogue during runtime. That estimate of the reward should ideally be obtained directly from the user. One way of achieving this is for the user to give a rating at the end of each dialogue. However, it has previously been shown that the perception of dialogue success varies greatly between different users [58]. Therefore, obtaining the reward directly from the user remains a challenge and will be discussed further in Chapter 8.

An alternative way to perform long-term supervised adaptation is to learn from a corpus of dialogues generated from the system's interaction with real users or a particular user. This requires some form of labelling. For instance, data labelled with the true user act can be used to re-estimate the observation and the transition probabilities of the POMDP dialogue model. This approach was explored in the Bayesian Update of Dialogue State system (see Section 2.5 and [73]), showing an improvement in performance. If the label of the true user act was given, the true user goal can be inferred. Based on that, the reward can be assigned to each dialogue turn [94]. This can then be used, for example, to estimate the kernel parameters of the Gaussian process model for the Q-function, as shown in Section 6.3.

Another important direction in adaptation concerns short-term, rapid adaptation. Such an approach to adaptation is useful in dialogue system applications where there is a small chance that the user will call again so performing long term adaptation to that particular user would not yield benefits. Instead, several policies can be trained off-line for different groups of users – *user types*. Then, the policy that is most appropriate for a particular user that the system is interacting with can be selected on-line. Such an approach enables adaptation within a single dialogue. The crucial element here is the policy selection criterion. If the reward estimate is given or if the true user action is known, short term adaptation is supervised. Then, the choice of the systems action can be based on the likelihood of the observed reward or the likelihood of the true user act. Another situation is when neither the reward nor the true user action are known. Then, adaptation is unsupervised. This adaptation approach will be examined further in this chapter where the Q-function is modelled as a Gaussian processes.

7.3 Policy selection based on the Gaussian process uncertainty estimate

The Gaussian process approximation for the Q-function presented in Section 6.2.5 provides an estimate of the mean and the variance of the Q-function for any belief state-action pair. The action that is chosen in the action-selection process is the one that is associated with the highest estimate of the mean for the current belief state.

If there are two Gaussian process models for the Q-function, trained for two different user types for example, then, for the same belief state, these two models can propose two different actions. The adaptation approach presented here advocates that the action that corresponds to the model which has a lower variance estimate for the Q-function in that action and the current belief state is selected.

This model selection criterion can be interpreted as choosing the model which is more certain about its estimate of the highest Q-value. The main advantage of this approach is that it only uses the estimate of the uncertainty that is already provided by the model to perform adaptation. This is in contrast to the adaptation approach that was taken in [141] where the dialogue state was extended to include user-type specific elements.

This approach has been inspired by adaptation in speech recognition, but it is important to note some major differences. In cluster adaptive training [142], models are trained on clusters, for example, speakers. Then, at run-time, after one pass of recognition, the model parameters for different clusters are interpolated. The interpolation weights are chosen to maximise the likelihood of the observation given the recognition hypothesis. These weights yield the resulting model, which is then used for recognition. In the case here, there are two Gaussian process models available, $\mathcal{M}_\mathcal{I}$ and $\mathcal{M}_\mathcal{E}$, which provide two different posterior estimates of the Q-function for any belief state action pair:

$$\mathcal{M}_\mathcal{I} : Q(\mathbf{b}, a) \sim \mathcal{GP}\left(\bar{Q}_\mathcal{I}(\mathbf{b}, a), cov_\mathcal{I}(\mathbf{b}, a, \mathbf{b}, a)\right), \tag{7.1}$$

$$\mathcal{M}_\mathcal{E} : Q(\mathbf{b}, a) \sim \mathcal{GP}\left(\bar{Q}_\mathcal{E}(\mathbf{b}, a), cov_\mathcal{E}(\mathbf{b}, a, \mathbf{b}, a)\right). \tag{7.2}$$

At run-time, for the current state $\mathbf{b}_t$, each model proposes the action that has the highest mean for that belief state (Eq. 6.11 in Section 6.2.1):

$$a_t^\mathcal{M} = \arg\max_a \bar{Q}_\mathcal{M}(\mathbf{b}_t, a), \tag{7.3}$$

where $\mathcal{M} \in \{\mathcal{M}_\mathcal{I}, \mathcal{M}_\mathcal{E}\}$. The likelihood of the mean given the model, $p(\bar{Q}_\mathcal{M}(\mathbf{b}_t, a_t^\mathcal{M})|\mathcal{M})$, only depends on the variance $cov_\mathcal{M}(\mathbf{b}_t, a_t^\mathcal{M}, \mathbf{b}_t, a_t^\mathcal{M})$. So selecting the model based on variance can be seen as selecting the model which has a higher likelihood for belief state $\mathbf{b}_t$ in turn t when the action is chosen to maximise the mean.

It is important to note that this is a heuristic. In fact, both action selection and model selection can be based solely on the mean or solely on the variance. However, in reinforcement learning, the action is chosen in such way as to maximise the expected reward. In other words, it is not chosen on the basis of how likely it is in a particular belief state but rather on the estimated return it can generate in a particular belief state. On the other hand, different models give different estimates of uncertainty for the optimal action in a particular belief state. Therefore, it makes intuitive sense to base the model selection criterion on the variance estimate.

7.4 Different user types

Users can be categorised in multiple ways. The simplest one is the distinction between a user that has had interaction with the system and a user that has not. It is common in human-computer dialogue that if the user has spoken to the system for long enough they learn which questions to ask in order to solicit the answer from the system more quickly. Conversely, first-time users typically have difficulties communicating their request to the system and such dialogues often end up being unsuccessful. Being able to adequately address both user categories is very important for successful dialogue operation [141].

As explained in Section 3.4.2, the agenda-based user simulator enables variability in behaviour, which is controlled with parameterised probability distributions. Varying these parameters enables a range of different user types to be created. Two different user types are examined here: one which rarely expresses the full goal and is generally less co-operative, the *inexperienced user*, and another which fully specifies the request, immediately corrects any mistake that the system made and fully answers the system's questions, the *experienced user*. The parameter settings for these user types are given in Table 7.1. In addition, the agenda-based simulated user incorporates a patience level. In the set-up used here, the user ends the dialogue once the system repeats the same action three times in a row.

In Table 7.2 an example dialogue with the experienced user is given. As can be seen from this example, the experienced user immediately specifies the full goal, hello(=coffeshop, pricerange=cheap, area=romsey) and changes their mind when needed as in reqalts(pricerange=dontcare). In Table 7.3 an example dialogue with the inexperienced user is given, where it is mainly the system that is asking for information and guiding the user, with dialogue acts such as request(task), request(area), reqmore().

7.5 Short-term adaptation evaluation results

To evaluate the proposed adaptation strategy, two Gaussian process models were trained, one with an inexperienced user another with an experienced user. The domain was the CamInfo task (Appendix C.1). The training set-ups for both models were the same as in Section 3.4.1 with the exception

Parameter	Default	Experienced	Inexperienced
InformCombination	0.600	0.900	0.100
AddAttributeToReq	0.333	0.666	0.111
YesAfterReqmore	0.250	0.100	0.600
AffirmWithAgdItem	0.050	0.600	0.050
Greeting	0.500	0.700	0.300
ConstraintRelax	0.667	0.900	0.333
TellAboutChange	0.500	0.900	0.200
ByeOrStartOver	0.333	0.100	0.500
DealWithPending	0.500	0.900	0.300
AddEntityName	0.050	0.500	0.050
NoAttrWithDontcare	0.800	0.500	0.900
InformToConfirm	0.050	0.700	0.050
ReqAltsAfterEntRec1	0.143	0.143	0.143
ReqAltsAfterEntRec2	0.143	0.143	0.143
RequestResponce1	0.200	0.050	0.200
RequestResponce2	0.200	0.010	0.600
OverruleCorrection	0.100	0.010	0.700
CorrectingAct1	0.450	0.700	0.200
CorrectingAct2	0.400	0.200	0.100
ThankAck1	0.100	0.100	0.600
ThankAck2	0.100	0.100	0.300

Table 7.1: Parameter settings for experienced and inexperienced user types.

that the maximum reward was set to be +100. The reason for this is that the interaction with the inexperienced user typically takes a larger number of dialogue turns than with the experienced user. The increased reward enables the system to engage in longer dialogues. The GP-Sarsa algorithm used the linear kernel and the parameter setting was the same as in Section 6.4.1. Training was performed in incremental noise (see Section 3.4.3) generated with the uniform error model (see Section 3.4.2).

Based on the policy selection criterion presented in Section 7.3, the policy can be selected either at the beginning of dialogue, as soon as the first input is received from the user, or after every dialogue turn. Both of these strategies are examined here. The baseline strategy is just a random selection between the two policies, again either at the beginning of the dialogue or after every dialogue turn.

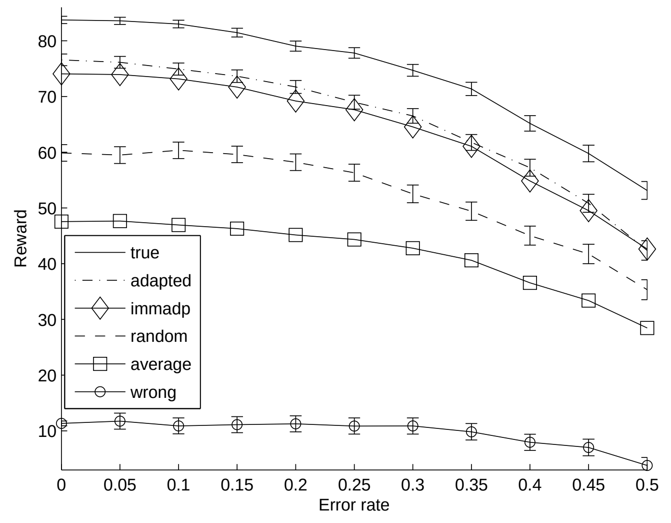
Referring firstly to Fig. 7.1 which shows performance for the inexperienced user, there were two trained policies, one trained on the experienced

System	hello()
User	hello(=coffeeshop,pricerange=cheap,area=romsey)
System	confirm(drinktype=coffeeshop,area=romsey)
User	affirm()
System	inform(name=none, type=placetodrink, drinktype=coffeeshop, area=romsey, pricerange=cheap)
User	reqalts(pricerange=dontcare)
System	inform(name="Cafe Adriatic", type=placetodrink, drinktype=coffeeshop, area=romsey, pricerange=expensive)
Usrer	ack()
System	inform(name="Cafe Adriatic",type=placetodrink, drinktype=coffeeshop, pricerange=expensive, area=romsey)
User	bye()

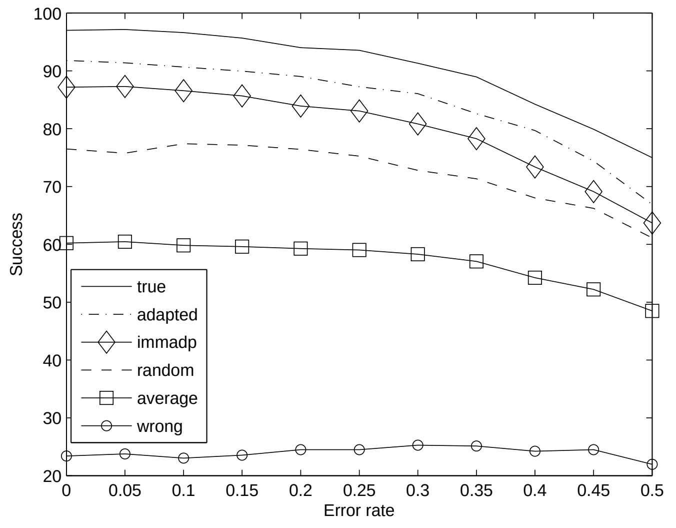
Table 7.2: Dialogue with an experienced user.

System	hello()
User	hello()
System	request(task)
User	inform(=museum)
System	request(area)
User	inform(area=fenditton)
System	inform(name="Taxi Gallery", type=placetosee, seetype=museum, area=fenditton)
User	inform(pricerange=dontcare)
System	reqmore()
User	request(addr)
System	inform(name="Taxi Gallery", addr="38 Stanesfield Road")
User	thankyou()
System	reqmore()
User	request(phone)
System	inform(name="Taxi Gallery", phone="01223576017")
User	ack()
System	reqmore()
User	bye()

Table 7.3: Dialogue with an inexperienced user.

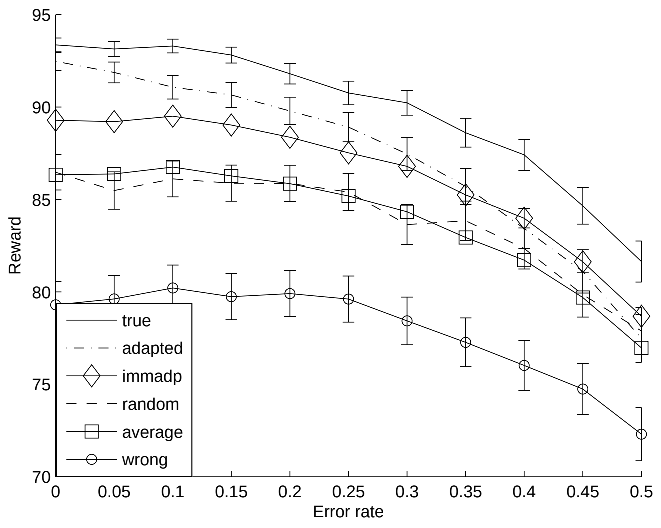


(a) Adaptation to inexperienced user, reward

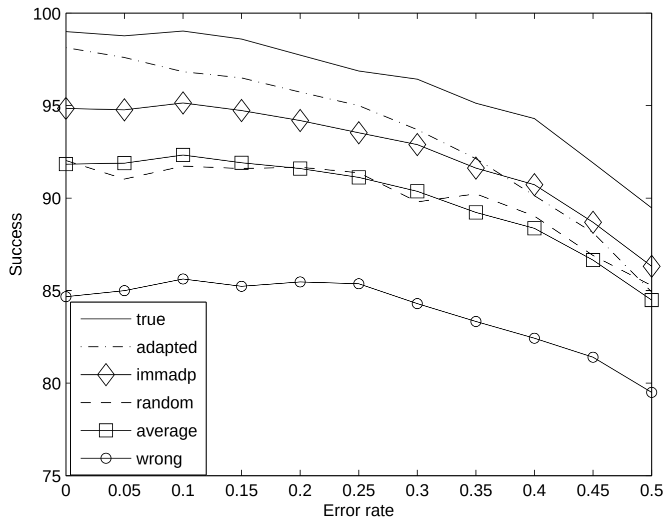


(b) Adaptation to inexperienced user, success

Figure 7.1: Adaptation to inexperienced user, reward (a) and success (b).



(a) Adaptation to experienced user, reward



(b) Adaptation to experienced user, success

Figure 7.2: Adaptation to experienced user, reward (a) and success (b).

user (wrong) and the other on the inexperienced user (true). There were two random selections one which chooses a policy at the beginning of each dialogue (average) and the other which chooses a policy at every dialogue turn (random). Finally, two adaptation strategies were considered, one which adapts at the beginning of the dialogue (immadp) and another which adapts at every dialogue turn (adapt). Fig. 7.2 shows the performance for the experienced user with the roles of the two user types reversed.

As expected, the matching policies, *i.e.*, the policies trained and tested on the same user type perform best (denoted as true on Figs. 7.1 and 7.2), while the mismatched policies, the policies trained on one user but tested on the other (denoted as wrong) have the worst performance.

The results for the mismatched policy reveal significantly worse performance when tested on the inexperienced user compared to the experienced user, the difference between wrong and true in Fig. 7.1 and Fig. 7.2. Corpus examination of the inexperienced user data shows that the majority of dialogues end with the user losing patience. The reason for this is that the inexperienced user is less co-operative than the experienced user. This can be seen from the different settings for the parameters OverruleCorrection and InformCombination in Table 7.1. Therefore, it is very difficult for the experienced user policy to navigate the inexperienced user to the part of the summary space where the user goal is defined, which is a prerequisite for the dialogue to be successful. On the other hand, the policy trained with the inexperienced user had visited parts of the space where the user goal is not defined and is able to deal with such a situation. Conversely, the fact that the experienced user is very likely to define the goal (see the InformCombination parameter value) is an advantage as it quickly reaches the part of the belief state space where the user goal is fully defined. In that case even a policy trained on the inexperienced user type has a high chance of success. Therefore, there is not such a dramatic drop in performance when the inexperienced user policy is tested on the experienced user.

Whether the random selection between different policies is performed at the turn level (random) or on the dialogue level (average) makes a significant difference when testing on the inexperienced user (Fig. 7.1). The reason for this lies mainly in the patience level. If the policy is chosen randomly at the beginning of the dialogue then there is a 50% chance of choosing the wrong policy. If the policy is chosen randomly at the turn level there is a

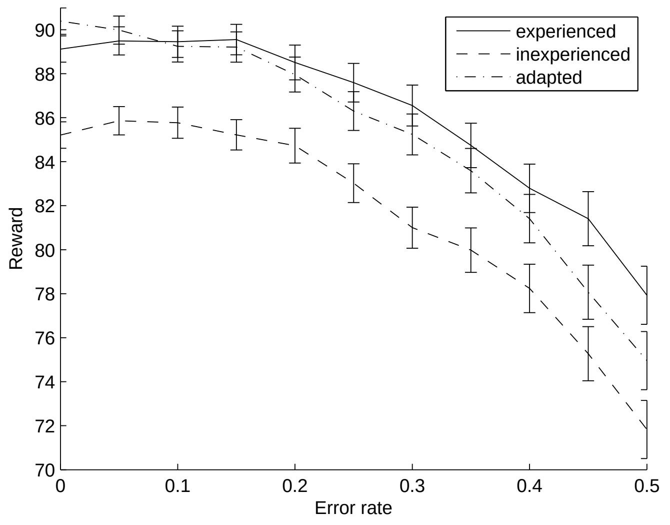
50% chance that the system will take a correct action which then reduces the possibility of the system repeating the same action three times consecutively and the user losing patience.

The difference between performing adaptation on the turn level (adapted) versus the dialogue level (immadp) is greatest when adapting to the experienced user (Fig. 7.2). There, turn-level adaptation reaches close to optimal performance at zero error rate. The only case when the uncertainty-based adaptation does not perform well is when both of the users visit the same part of the space frequently and have different optimal actions associated with that part of the space. In the case of the experienced user in the overlapping parts of the space, the inexperienced user policy proposes the optimal action. In addition, it manages to detect the user type correctly in other parts of the space so it achieves close to optimal performance. With an increase in noise level, however, it is harder to detect the user type so adaptation performance degrades.

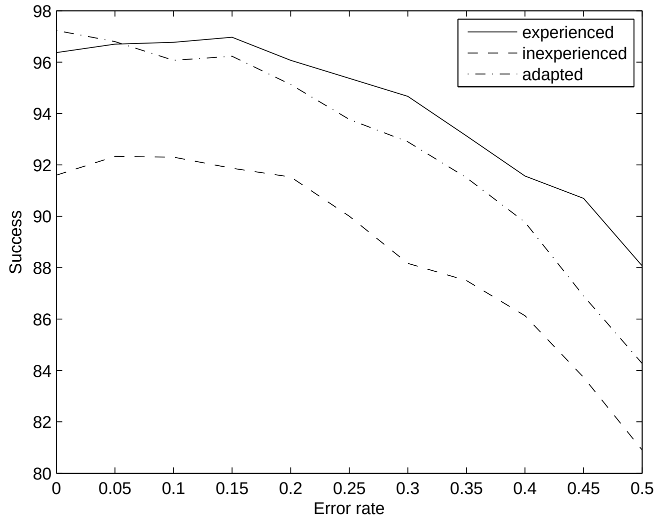
The experienced and inexperienced user types may be regarded as somewhat extreme since they have been deliberately configured to be distinct. To see whether or not the policies trained on these user types are in any way atypical, they were also tested on the default user type (used elsewhere in the thesis). The results are shown in Fig. 7.3. Both policies provide reasonable performance, although the experienced policy is the best suggesting that the default user type was more similar to the experienced user than the inexperienced user. Note also that adaptation also works reasonably well for the default user, especially at low error rates.

7.6 Training on both user types

The previous section has shown that the proposed adaptation strategy is able to perform well. However, in order to examine the full value of adaptation, the question that is posed is whether training a policy on both users yields a good policy too or whether it is necessary to separate the user types during the training and then adapt during the testing. This is directly related to the differences between the experienced user policy and the inexperienced user policy. There are two ways in which trained policies can behave differently. Firstly, the policies may be trained in different parts of the state-action space. Secondly, they may cover the same part of the space, but they have



(a) Evaluation on the default user type, reward



(b) Evaluation on the default user, success

Figure 7.3: Evaluation on the default user, reward (a) and success (b).

differing optimal actions in some states.

The experiments in this section compare the matching policy, the short-term adaptation strategy from Section 7.3 and the policy that is trained on both user types. The latter is a policy that is trained by choosing the user type randomly at the beginning of each dialogue.

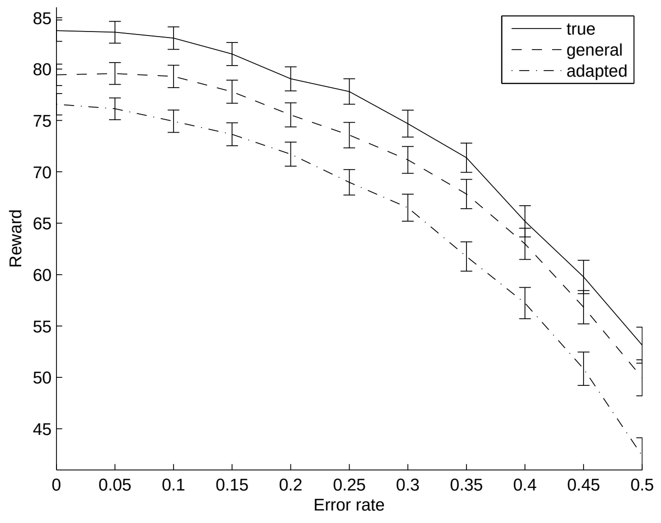
The evaluation results are given in Fig. 7.4 and Fig. 7.5. The results show that the policy trained on both user types generalises well for the experienced user (see Fig. 7.5). For the inexperienced user, the average reward is lower (see Fig. 7.4(a)), but the average success is indistinguishable (see Fig. 7.4(b)). In both cases it outperforms the adaptation strategy.

In order to explain this generalisation capability there are two aspects that require further inspection. The first question is how different the Q-estimates are for different user types in the same parts of the summary space; the second question is how often different user types visit the same parts of the summary space. GP-Sarsa defines a probability distribution for every state-action pair, which makes it hard to directly investigate these issues. Therefore, the ability of the policy trained on both user types to generalise well was investigated using the Monte Carlo control (MCC) algorithm. As already explained in Section 3.4.1, the Monte Carlo control algorithm gathers visited grid-points, the summary points, and estimates the Q-function in these points for each action and has an associated frequency of how often that state action pair is visited. This makes it easier to examine the aforementioned issues.

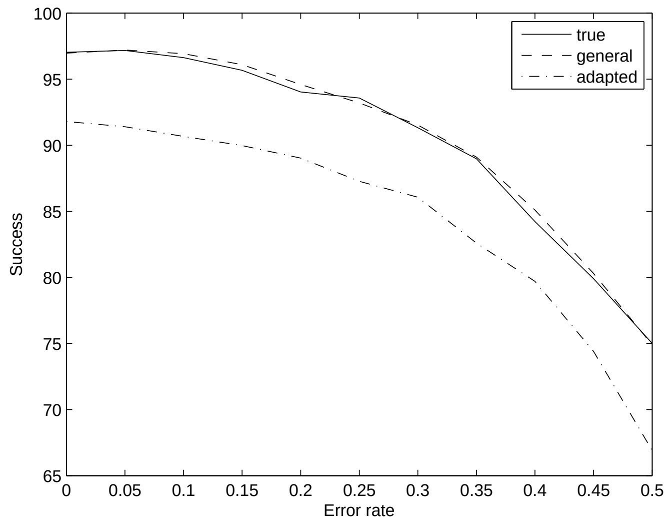
The Monte Carlo control algorithm was used to train policies with the experienced user, the inexperienced user and a policy for both user types. The training set-up is the same as in Section 6.4.2 except that the maximum reward given was +100 and the that training was performed in incremental noise (see Section 3.4.3) generated with the uniform error model (see Section 3.4.2).

	MCC grid points	GP-Sarsa dictionary points
Policy trained on experienced user	124	578
Policy trained on inexperienced user	131	713
Policy trained on both users	127	710

Table 7.4: Comparison of the number of policy points for different user types.

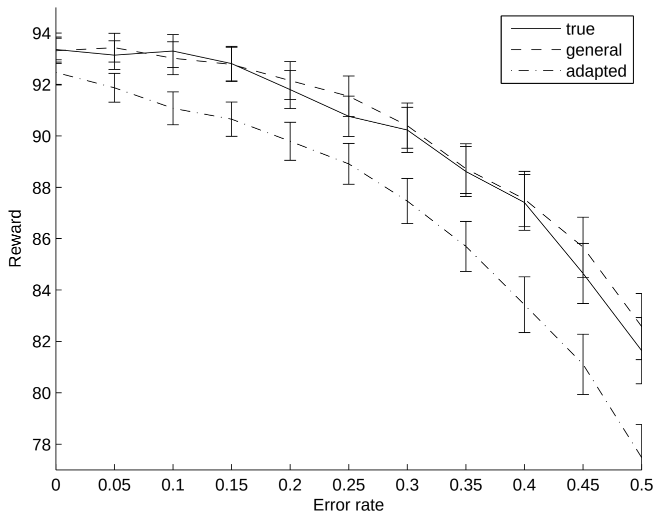


(a) Generalisation to the inexperienced user, reward

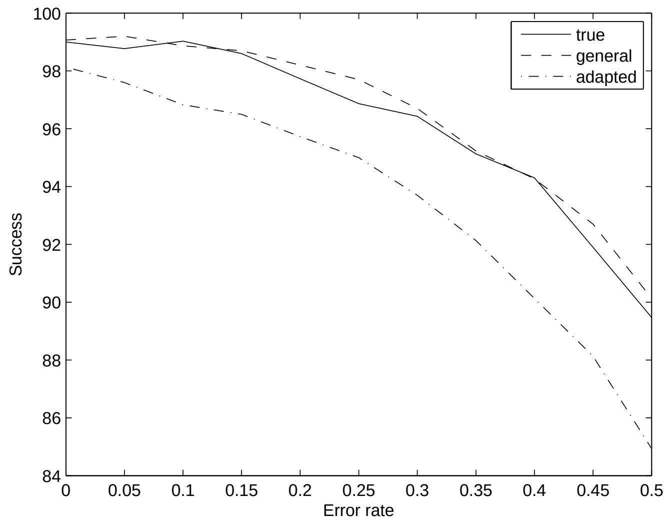


(b) Generalisation to the inexperienced user, success

Figure 7.4: Generalisation to inexperienced user, reward (a) and success (b).



(a) Generalisation to the experienced user, reward



(b) Generalisation to the experienced user, success

Figure 7.5: Generalisation to experienced user, reward (a) and success (b).

Table 7.4 gives a comparison of the number of points that the policies for different user types consist of for both the Monte Carlo control algorithm and the GP-Sarsa algorithm. Although the parameter ν that controls the number of points was the same for all policies, it is important to note that the number of GP-Sarsa points and the number of MCC points are not directly comparable since they have different interpretations in these algorithms. However, this table does give some insight into the differences between the different user types. In the case of the GP-Sarsa algorithm, the policy generated with the experienced user type has fewer representative points than the policy trained on inexperienced user type or the policy trained on both user types. This means that the experienced user covers a smaller part of the space than the other user types. However, the number of grid-points that the Monte Carlo control algorithm generates for the experienced user type is not much smaller than for the other user types, which suggests that the grid that this algorithm produces is relatively crude.

Closer examination of the Monte Carlo control grid points for different user types revealed that they mostly coincide in the discrete features. The discrete part of the summary space is non-Markovian and it roughly corresponds to a dialogue turn. Since both user types generate the same dialogue acts with varying probabilities, both of the policies' summary points cover the same discrete part of the summary space. However, the frequencies with which different parts of the summary space are visited vary and this leads to different Q-function estimates for these two user types. The estimates are not trained well in places where frequencies are low. Conversely, for the policy trained on both user types, these frequencies are high for all summary points. In principle, the estimate of the expected discounted reward for the same part of the summary state is different for the experienced and the inexperienced user. However, the reward function is fairly simplistic and only incorporates the notion of success, which is the same for both users, and the number of dialogue turns. Therefore, the estimate of the expected discounted reward is consistently higher for the experienced user than the inexperienced user in the same part of the summary space. So while the rewards are different, in every part of the space there exists an optimal action which is the same for both user types. Thus, there is no conflicting behaviour which would prevent the policy trained on both users to generalise well.

In Tables 7.5 and 7.6, the performance of the Monte Carlo control algorithm and that of the GP-Sarsa algorithm are directly compared. The policies were trained on the experienced user, the inexperienced user and both user types. The adaptation strategy that was examined in the MCC algorithm is simply to choose the point from the two policies that is the closest to the current point at run-time and is performed on the dialogue turn level. The adaptation strategy for GP-Sarsa is uncertainty-based as explained in Section 7.3 and is also performed on the dialogue turn level. This is only a preliminary comparison. It is performed at zero error rate to avoid the confusion that may arise from higher error rates thereby masking the effects of adaptation.

Results on the inexperienced user (Table 7.5) show that both algorithms achieved similar performance when trained on the inexperienced user type. Also, the policies trained on both user types and tested on the inexperienced user type (general) perform similarly for both algorithms. However, the adaptation capability of GP-Sarsa is much greater than the Monte Carlo control algorithm's. The adaptation strategy of MCC does not take into account how often each part of the space was visited but simply approximates the current point with the closest point in each of the two separately trained policies. While MCC can be heuristically modified to take into account the frequencies of grid-point occupation during training, GP-Sarsa incorporates this information elegantly via a Bayesian approach which maintains a Gaussian distribution over the Q-function for every part of the continuous space (see Eq. 6.10 in Section 6.2.2).

On the other hand, the performance of the Monte Carlo control algorithm is better than the GP-Sarsa when testing on mis-matching user types. The grid that the MCC generates is very crude and dilutes the effect of differences between the user types. As already discussed above, modelling these differences is not essential for good policy performance due to the lack of conflicting rewards. Therefore, a crude grid can be seen as a generalisation across the continuous space which leads to a better performance on different users.

These effects are not as dramatic when adapting to the experienced user (Table 7.6), however they are still noticeable, particularly in the measure of dialogue success.

	Reward (std. dev.)		Success	
	MCC	GP-Sarsa	MCC	GP-Sarsa
Matching policy	82.76 (0.79)	83.74 (0.66)	96.10	97.03
Policy trained on both	82.04 (0.83)	79.43 (0.69)	95.90	96.97
Adaptation	58.65 (1.58)	**76.58** (**1.04**)	71.87	**91.80**
Mismatched policy	**28.65** (**1.69**)	11.35 (0.43)	**41.60**	23.40

Table 7.5: Comparison of the MCC and the GP-Sarsa on adaptation to the inexperienced user type, zero error rate.

	Reward (std. dev.)		Success	
	MCC	GP-Sarsa	MCC	GP-Sarsa
Matching policy	94.39 (0.25)	93.36 (0.37)	99.53	99.00
Policy trained on both	93.18 (0.41)	93.31 (0.36)	98.80	99.07
Adaptation	89.21 (0.78)	92.47 (0.49)	95.03	**98.13**
Mismatched policy	81.87 (1.17)	79.30 (1.27)	**87.50**	84.67

Table 7.6: Comparison of the MCC and the GP-Sarsa on adaptation to the experienced user type, zero error rate.

7.7 Summary

In this chapter, a short-term adaptation strategy to different user types has been examined. The key idea of the strategy is to select the policy associated with the GP model that has the lowest uncertainty for the optimal action. This adaptation strategy was evaluated using the HIS system in interaction with an agenda-based user simulator which was parameterised to simulate different user types.

The results have shown that the strategy significantly outperforms random policy selection and also an alternative grid-based adaptation scheme, demonstrating the capability of the Gaussian process model of the Q-function to detect different user types.

However, the adaptation strategy was out-performed by a simple policy trained on both user types. The reason why this policy generalises so well across different user types is probably because the HIS summary space does not preserve the Markov property of the full belief space. This results in an overlap between those parts of the summary space visited by the different user types. In addition, for the same parts of the summary space, the estimate of the expected reward is consistently higher for the experienced user than for the inexperienced user. There therefore exists an optimal Q-function that is the same for both users. This explains why little discrimination between the users' behaviour is necessary to obtain a near optimal policy. This is clearly a weakness of the HIS summary space and it may be expected that when more discriminating statistical dialogue models are developed, the full value of the Gaussian process approach for unsupervised adaptation will become evident.

Chapter 8

Conclusion

8.1 Thesis summary

This thesis has examined the challenges that present themselves when applying a partially observable Markov decision process-based dialogue manager to a real-world task in the context of the Hidden Information State (HIS) framework. Adopting a POMDP-based approach requires that a distribution over all dialogue states, the *belief state*, is maintained through the dialogue. This process is called the *belief update.* Action selection is based on the belief state and is determined by the dialogue manager policy. The process of learning the optimal policy is called *policy optimisation.* This thesis has proposed a more efficient representation of the dialogue state. It has also examined two aspects of dialogue policy optimisation. The first concerns the problems that arise when learning in a reduced summary space of the belief state space. The second proposes using Gaussian processes for reinforcement learning to achieve fast policy optimisation and to enable adaptation to different user types.

One of the main limiting factors that inhibits the application of POMDP-based dialogue management to problems with a large number of states is the intractability of the belief update and policy optimisation. The Hidden Information State dialogue manager deals with the huge number of possible dialogue states by grouping them together in *partitions*, elements dynamically built from the concepts in the user input. This reduces the computational cost of performing the belief update. It does not, however, address the problem of partitions' exponential growth with the number of concepts in the N-best list of user inputs. This places a constraint on the length of the N-best recognition hypotheses input from the recogniser and the length of the dialogue that can be supported. This thesis has proposed an alternative partition representation that depends on maintaining the logical complements of the concepts expressed by the alternative user inputs (Section 4.2). This allows logical expressions to be used both by the user and the system, thus supporting more complex dialogues (Section 4.4). More importantly, the alternative partition representation supports a pruning technique which removes the constraint on the length of the N-best list and the number of dialogue turns that can be supported (Section 4.7.2). It has been shown in this thesis that this pruning technique represents a significant improvement on the original HIS branching-tree representation and furthermore that it outperforms partition recombination [110], an alternative pruning method, when assessed on a simulated user (Section 4.8).

This thesis has examined policy optimisation for dialogue management in two areas. Firstly, the problems that occur when optimising a policy in a reduced *summary space* rather than the full *master space* have been examined. Secondly, Bayesian methods of policy optimisation were investigated in order to achieve faster convergence.

In reinforcement learning the optimal policy is derived from the Q-function which is the expected discounted reward for a belief state action pair. It is a common approach in POMDP policy optimisation to reduce the space of all belief states in order to achieve tractability in learning. However, a consequence of this reduction is that mapping a selected summary action back to the master space is not always possible. This thesis has investigated this problem and has proposed the use of the Q-function values to sort actions in an N-best list of possible back-off actions (Section 5.4). It was shown that such a method is guaranteed to give the highest expected performance (Section 5.2) and this was verified empirically in interaction with a simulated user (Section 5.5). Additionally, it has been found that extending the summary space drastically reduces the robustness of the resulting policy. This shows that standard non-parametric approaches to policy optimisation place a constraint on the size of the summary space and confirms the need for a faster policy optimisation technique.

Policy optimisation was examined in the more general context of achieving faster learning without the use of domain-specific approximations. It proposes the use of Gaussian processes in reinforcement leaning as a non-parametric, Bayesian method for estimating the Q-function. Previous work on non-parametric approaches to policy optimisation used a heavily reduced summary space, and then discretised this space into a grid for which conventional reinforcement learning algorithms are tractable. This approach has been taken in the HIS system. While standard non-parametric approaches, such as the grid-based approach, achieve tractability, they are limited to cases where the summary space is very small and they depend on the use of a crude grid, which inevitably leads to sub-optimal results. An alternative is to use a parametric approach. In the Bayesian Update of Dialogue State (BUDS) system [73], see Section 2.2.6, the policy is defined as a parameterised combination of basis functions and the optimal solution is found with gradient methods. There are several issues to be addressed in the use of the parametric approach. Firstly, it requires the set of basis function to be set by hand, using expert knowledge of the domain. Secondly, parametric approaches are only guaranteed to give the optimal solution within the given basis, so if the basis is not chosen correctly the solution will be suboptimal. This thesis has investigated modelling the Q-function as a Gaussian process (Section 6.2.2).

Gaussian processes incorporate knowledge of correlations in function values in different parts of the space via the *kernel function*. If the kernel function is chosen correctly, the optimal solution can be discovered more quickly than by standard non-parametric methods, thus speeding up the overall process of policy optimisation. Modelling the Q-function as a Gaussian process has thus the capability of making the process of finding the optimal dialogue policy faster. However, to compute the Gaussian process posterior tractably, a sparsification method must be used.

The kernel span sparsification method used here (Section 6.2.4) has the effect of transforming a non-parametric method into a parametric method. Nevertheless, on a toy dialogue problem examined in this thesis, this method demonstrated faster convergence times than a standard reinforcement learning algorithm, while reaching a close to optimal performance (Section 6.3.3). It has furthermore been shown that the kernel parameters can be learnt from data annotated with rewards, actions and belief states, without the need to annotate the true underlying dialogue state. Modelling the Q-function as a Gaussian process has also been investigated in the context of a real-world problem (Section 6.4). It achieved a much faster convergence than the Monte Carlo control algorithm on the simulated user. In addition, the incorporation of active learning using the estimate of the variance of the posterior to determine which actions should be explored in ϵ-greedy reinforcement learning led to a further acceleration of the policy optimisation process on the simulated user, obtaining more than 85% success rate after only 200 dialogues. This makes it feasible to perform learning in direct interaction with humans.

Another use of the measure of uncertainty has been examined in this thesis within the context of adaptation to different user types obtained by varying parameters of the agenda-based user simulator (Section 3.4.2). This preliminary work has shown that the posterior variance that is obtained by modelling the Q-function as a Gaussian process can be used to detect the user type and upon that select the appropriate policy to use (Section 7.5). While this adaptation strategy did not perform as well as a universal policy trained on all user types, it was still able to detect the correct user type after the first turn at least 70% of the time, and that percentage stays roughly the same even at high noise levels. Furthermore, training a single universal policy will become intractable as systems become more complex and the user population becomes larger and more diverse. When this occurs, the performance of systems able to perform robust user-dependent policy selection should outperform those relying on a single generalised policy.

The work presented in this thesis has also shed light on the main limitations of the studied framework as well as in the methods that have been proposed in this thesis. These limitations are discussed in the following section. In Section 8.3 further research proposals are advanced that will address these issues to facilitate full adaptive systems that can ultimately be deployed in on-line learning with real users.

8.2 Limitations

All the methods presented in this thesis have been implemented within the Hidden Information State framework. While this framework facilitates the application of the POMDP approach to real-world dialogue tasks, it has several limitations, some of which have been addressed in the thesis. The problem of the exponential growth of partitions during dialogue has been fully investigated in this thesis. However, the problem of users' changing goal has only been partially investigated. This problem originates mainly in the assumption made in the HIS framework that the user does not change their mind during belief update. To address this issue adequately, the belief update formula would need to incorporate the possibility of the user goal's changing. Another issue is that the user model component is not easily parameterised and this limits its potential for learning the finer aspects of user behaviour from data. The approach taken in the Bayesian Update of Dialogue State (BUDS) framework is more flexible in that respect and enables more elements of the user model to be learnt from human-computer data.

There are currently several hand-crafted components in the Hidden Information State framework that cannot be derived from data. These include the domain ontology, the dialogue act specification, the dialogue act matching model and the dialogue history model, but the most limiting of all is the summary space. Not only is it based on heuristics, but it also constrains the scope of system behaviour that can be automatically optimised. The idea behind the use of Gaussian processes in reinforcement learning for dialogue management is completely to avoid the use of the summary space and to define correlations directly on the full belief space. However, defining the kernel function directly on the full belief space is non-trivial.

This thesis has examined a simple form of unsupervised adaptation that uses the uncertainty measure the Gaussian process reinforcement learning provides. While the proposed adaptation method performed better than the random policy selection, it was not always able to detect the correct user type. This may be attributed to the unreliable estimate of uncertainty. More precisely, while the examined sparsification method does achieve tractability in Gaussian process learning, it has been shown to give overly confident estimates. This is not desirable for the proposed adaptation strategy. In addition, the kernel span sparsification method turns a non-parametric method, which limits the convergence properties of the approximated Gaussian process, so an alternative sparsification method is needed.

In order fully to exploit the potential of reinforcement learning to perform direct on-line learning for real users the reward is a crucial element and the problem of obtaining it directly from the user poses a major obstacle.

Finally, all the results presented in this thesis have been obtained in interaction with the agenda-based simulated user. While the simulated user enables a variety of dialogues to be generated, its behaviour is more constrained than is real user behaviour. Accordingly, the full value of these methods needs to be confirmed in evaluation with real users.

8.3 Future work

This section presents research proposals that need to be pursued in order to resolve the limitations that currently prevent POMDP-based dialogue management from being fully adaptable.

This thesis has shown that applying Gaussian processes in reinforcement learning significantly speeds up policy optimisation. An active learning method brought further improvements. It simply chooses the action based on the uncertainty estimate with probability ϵ during learning. The problem with ϵ-greedy learning is that ϵ is set heuristically. In order to avoid this, an alternative active learning method can be used where instead of choosing the action based on the variance of the Q-function during exploration and using the mean of the Q-function during exploitation, one can simply sample Q-function values throughout the learning process. This is similar to using a stochastic policy [32]. A similar approach was taken in the BUDS system where a parametrised stochastic policy is used.

While the results of show that GP-Sarsa is faster in convergence compared to the Monte Carlo control algorithm, the summary space that the dialogue manager operates on is very limited. The framework, however, allows the learning to take place on the full belief space. This requires the kernel function to be defined on the belief state space and dialogue action space. The kernel on the belief space can be composed of a kernel on the partition space, a kernel on the user action space and a kernel function on the dialogue history space. The kernel function on the user action space and the system action space can be the same since both user and system actions follow the same dialogue act structure (see Table A.1 in Appendix A). Elements of these spaces have variable length and the kernel functions would have to be able to support this. An example of such a kernel is the rational kernel [137]. Due to the partition structure the kernel defined for partition space has to operate on trees and one such kernel is given in [136]. Further investigation of these kernels will be part of future work.

If these kernel functions were defined, it would also allow Gaussian processes to be applied to the user goal model, the user action model and the dialogue history model (see Section 2.2.6). These elements are mostly hand coded in the HIS system (see Section 3.3.4) and limit its flexibility. Gaussian processes have also been applied to model the transition probabilities of a continuous MDP [118] and they have also been applied to model the transition and observation probabilities in a POMDP [143]. It remains for future work to apply this to POMDP-based spoken dialogue management. If successful, this would be an alternative to the BUDS system, which must make a number of approximating conditional independence assumptions in order to achieve tractable belief monitoring and estimation of the transition probabilities from data.

As already noted, the sparsification method used to enable tractable update in Gaussian process-based reinforcement learning produces overly-confident estimates. In order to ensure a more realistic estimate of the variance of the posterior, while preserving tractability at the same time, a better sparse approximation of Gaussian processes is required. The Fully Independent Training Conditional (FITC) approximation has been shown to have the desired properties, but it requires a small set of the support points to be obtained (see Section 6.2.4 and [123, 124]). One way to achieve this on-line would be to use the kernel span sparsification method to generate these points on-line. Then, instead of approximating the kernel as an inner product of feature vectors in these points as described in Section 6.2.4 and [125], these points can be used as support points in FITC. More precisely, the Q-function values can be assumed to be conditionally independent given the set of support points. This results in a process which has the same computational cost as the kernel span sparsification method, but provides a better estimate of the variance of the posterior.

Another matter for future investigation is estimation of the kernel function from data. The choice of the kernel function is crucial to successful application of Gaussian processes. The hyper-parameters could be optimised on a real dialogue corpus labelled with rewards. Natural gradient descent (Section 2.2.6) could then be used to find the hyper-parameters that maximise the marginal likelihood. The corpus may also be used to find the set of support points off-line, needed for the FITC approximation mentioned above. Splitting the corpus according to user profile would allow the hyper-parameters to be optimised for each particular user type separately. That would provide a better Gaussian process model of the Q-function for each user type and would give improvements in adaptation.

A critical element needed for on-line reinforcement learning is the estimate of reward. Ideally this should be obtained directly from the user. If an estimate of the reward during runtime was available, this would enable better adaptation techniques where the model improves in direct interaction with the real users. One way to obtain the reward from the real user might be to use emotion recognition. However, state-of-the art emotion recognition systems are not yet robust enough [144]. In dialogue systems the input is often corrupted with noise, so the emotion recogniser must be able to provide a robust estimate. Further investigation would yield benefits.

Finally, as already mentioned, all methods presented in this thesis have been trained and evaluated on a simulated user. While testing on real users would clearly be better, the cost of running sufficient user trials has hitherto been prohibitive. However, recent developments in crowd-sourcing have greatly simplified the problems of recruiting large numbers of subjects willing to perform prescribed tasks for a very small fee. This new source of live testing will in future make it possible to collect large amounts of real data simply and cheaply and this will significantly enhance our ability to conduct realistic experiments and make progress in the field [145].

Appendix A

Dialogue act formalism

In Table A.1 CUED dialogue act definitions are given. First column represents the dialogue act, second and third column denote whether the dialogue act is applicable to the system, the user or both and in the final column the dialogue act description is given.

Act	System	User	Description
hello()	✓	✓	start dialogue
hello(a=x,b=y, . . .)	×	✓	start dialogue and give information a=x, b=y, ...
silence()	×	✓	the user was silent
thankyou()	×	✓	non-specifying positive answer from the user
ack()	×	✓	the user acknowledged the system's response
bye()	✓	✓	end dialogue
hangup()	×	✓	user hangs up
inform(a=x, b=y, . . .)	✓	✓	give information a=x, b=y, ...
inform(name=none)	✓	×	inform that no suitable entity can be found
inform(a!=x, . . .)	×	✓	inform that a is not equal to x
inform(a=dontcare, . . .)	×	✓	the user does not care about the value of a
request(a)	✓	✓	request value of a
request(a, b=x, . . .)	✓	✓	request value of a given b=x,...
reqalts()	×	✓	request alternative solution
reqalts(a=x, . . .)	×	✓	request alternative solution with a=x,...
reqalts(a=dontcare, . . .)	×	✓	request alternative solution relaxing constraint a
reqmore()	✓	×	inquire if user wants anything more
reqmore()	×	✓	request more information about the current solution
reqmore(a=dontcare)	✓	×	inquire if user would like to relax a
reqmore(a=x,b=y, . . .)	×	✓	request more information given a=x, b=y, ...
confirm(a=x,b=y, . . .)	✓	✓	confirm a=x, b=y, ...
confirm(a!=x, . . .)	✓	✓	confirm a!=x, ...
confirm(name=none)	×	✓	confirm that no suitable entity can be found
confreq(a=x,...,c=z,d)	✓	×	confirm a=x, ... , c=z and request value of d
select(a=x, a=y)	✓	×	select either a=x or a=y
affirm()	✓	✓	simple yes response
affirm(a=x,b=y, . . .)	✓	✓	affirm and give further info a=x, b=y, ...
negate()	✓	✓	simple no response
negate(a=x)	✓	✓	negate and give corrected value for a
negate(a=x,b=y, . . .)	✓	✓	negate(a=x) and give further info b=y, ...
deny(a=x,b=y)	×	✓	no, a!=x and give further info b=y, ...
repeat()	✓	✓	request to repeat last act
help()	×	✓	request for help
restart()	×	✓	request to restart
null()	✓	✓	null act does nothing

Table A.1: Dialogue acts.

Appendix B

Pruning

B.1 Consistency proofs for pruning operations

Definition 1. *An ontology is a forest of ordered trees such that all its trees have the same root node and if a tree contains a node that has multiple child nodes, then the first child node determines the other child nodes of that node.*

Definition 2. *An attribute-value pair is a pair of nodes from some ontology tree where the value is a leaf node and the attribute is its parent node. It is denoted as $\alpha = \beta$.*

Definition 3. *Attribute-value pair $\alpha = \beta$ is called superior to attribute-value pair $\alpha' = \beta'$ if, in some ontology tree, value β is the first child node of node α and among other child nodes of α, $\beta_1, \ldots, \beta_n$, there exists node β_i so that nodes α and β_i appear on the path from α' to the root in that ontology tree.*

Definition 4. *A partition is a tree where all its non-leaf nodes coincide with all non-leaf nodes of at least one ontology tree, the coinciding ontology tree. Each leaf node of a partition is a set of boolean indicators. There is* *one indicator for each coinciding ontology tree. Each indicator corresponds to the value in an attribute-value pair in the coinciding ontology tree, where the attribute coincides with the parent of the leaf node in the partition.*

Definition 5. *The generic partition is the partition that has only two nodes. The root node of the generic partition coincides with the root node of every tree in the ontology and the leaf node has all indicators set to true.*

Definition 6. *Two partitions p and c are complementary in attribute-value pair $\alpha = \beta$ if they are the same in all nodes except in the nodes the child nodes of which have all indicators set to true in either p or c, and in the child node η of node α, where in partition p all boolean indicators in node η are set to false apart from the one for β, which is set to true, and in partition c the indicator for β in node η is set to false. It is said that partition p contains $\alpha = \beta$ and partition c contains $\alpha = \neg\beta$.*

Definition 7. *Attribute-value pair $\alpha = \beta$ is applicable to partition p if partition p contains the boolean indicator for value β set to true in the child node η of node α and at least one more indicator set to true in node η. Then, the process of applying attribute-value pair $\alpha = \beta$ to partition p is setting the indicator for β to false in partition p and replicating partition p into partition c where all indicators are set to false in node η, apart from the indicator for β, which is set to true. In the case where α has multiple child nodes in one of the coinciding ontology tree of partition c and β is its first child node in that ontology tree, other child nodes of node α are created in that partition. Every newly created child node then has a child leaf node where all indicators are set to true. The same applies to partition p if only one indicator in node η remains true. Partition c is called a child of partition p and partition p is called the parent of c. The process of applying* *attribute-value pair to a partition is also called partitioning.*

Statement 1. *Partition p and its child partition c created by partitioning p with $\alpha = \beta$ to p are complementary in $\alpha = \beta$.*

Proof. Trivial from Defs. 6 and 7. □

Definition 8. *A tree of partitions is an ordered tree where the nodes are partitions connected in parent-child relationship. The tree of partitions is created by applying ordered list of attribute-value pairs $\alpha_1 = \beta_1, \ldots, \alpha_n = \beta_n$ in such way that $\alpha_1 = \beta_1$ is applied to the generic partition, $\alpha_2 = \beta_2$ is applied to all resulting partitions and so on for every subsequent attribute-value pair from the list.*

Definition 9. *A subtree of partitions is a tree of partitions where instead of the generic partition the partitioning process starts with an arbitrary partition.*

Statement 2. *Two trees of partitions are the same if they are created using the same ordered list of attribute-value pairs.*

Proof. Trivial from Def. 8. □

Statement 3. *Two subtrees of partitions are the same if they are created using the same ordered list of attribute-value pairs starting from the same partition.*

Proof. Trivial from Stat. 2. □

Statement 4. *Attribute-value pair $\alpha = \beta$ cannot be applied to partition p if partition p does not contain all the superior attribute-value pairs of $\alpha = \beta$ that belong to one coinciding ontology tree of partition p.*

Proof. Trivial from Defs. 3 and 7. □

Statement 5. *Let T be the tree of partitions created with ordered list of attribute-value pairs $\alpha_1 = \beta_1, \dots, \alpha_k = \beta_k, \alpha = \beta, \alpha_{k+1} = \beta_{k+1}, \dots, \alpha_n = \beta_n$. Then, there exists at least one pair of partitions p and c in T such that p is the parent of c, they are complementary in $\alpha = \beta$ and for every such p there exists only one c.*

Proof. Let T' be the tree of partitions created by attribute-value pairs $\alpha_1 = \beta_1, \dots, \alpha_k = \beta_k$. Then, in order to apply attribute value pair $\alpha = \beta$, $\alpha = \beta$ has to be applicable to at least one partition p. Applying $\alpha = \beta$ to partition p creates partition c such that p is the parent of c and they are complementary in $\alpha = \beta$ (Def. 7 and Stat. 1).

Attribute-value pair $\alpha = \beta$ is only applied to partition p if it contains node η where the indicator for value β set to true. Once p is partitioned, the indicator is set to false in node η and therefore $\alpha = \beta$ cannot be reapplied to partition p (Def. 7). Hence, another such child partition c cannot be created. □

Definition 10. *Pruning of attribute-value pair $\alpha = \beta$ from tree of partitions T built using ordered list of attribute-value pairs $\alpha_1 = \beta_1, \dots, \alpha_k = \beta_k, \alpha = \beta, \alpha_{k+1} = \beta_{k+1}, \dots, \alpha_n = \beta_n$ is the process of removing all partitions from the tree T that are built using $\alpha = \beta$ or created from partitions that are built using $\alpha = \beta$ so that the resulting T' is the same as if it was built using attribute-value pairs $\alpha_1 = \beta_1, \dots, \alpha_k = \beta_k, \alpha_{k+1} = \beta_{k+1}, \dots, \alpha_n = \beta_n$.*

Statement 6. *Let T be the tree of partitions created by applying ordered list of attribute-value pairs $\alpha_1 = \beta_1, \dots, \alpha_k = \beta_k, \alpha = \beta, \alpha_{k+1} = \beta_{k+1}, \dots, \alpha_n = \beta_n$. Let p and c be a pair of partitions such that p is the parent of c and* *they are complementary in $\alpha = \beta$. Let $\alpha = \beta$ be not among the superiors for any attribute-value pair $\alpha_i = \beta_i$, $i = k+1, \ldots, n$. If p and c are roots of subtrees, these subtrees have the same structure and for every partition in one subtree there exists a partition in the same place in the other subtree and such that they are complementary in $\alpha = \beta$.*

Proof. Assume that every attribute-value pair $\alpha_i = \beta_i$ from $\alpha_{k+1} = \beta_{k+1}, \ldots, \alpha_n = \beta_n$, if applicable to a partition that contains $\alpha = \beta$, is then also applicable to partition that contains $\alpha = \neg\beta$. Then, sublist of applied attribute-value pairs $\alpha_{i_1} = \beta_{i_1}, \ldots, \alpha_{i_m} = \beta_{i_m}$ that is applicable to a partition which contains $\alpha = \beta$, is also applicable to a partition which contains $\alpha = \neg\beta$. By Defs. 7 and 6 partition p contains $\alpha = \beta$ and partition c contains $\alpha = \neg\beta$. Since the sublist $\alpha_{i_1} = \beta_{i_1}, \ldots, \alpha_{i_m} = \beta_{i_m}$ is applied in the same order to both of these partitions, it creates two subtrees with roots p and c respectively. These subtrees have the same structure since the list is applied in the same order and by Def. 8 a tree of partitions is an ordered tree. Had p and c been identical then these subtrees would have been identical too (Stat. 3). Since partitions p and c only differ in the child node η of node α, every partition of one subtree only differs in node η from the partition in the same position in the other subtree. In that way, for every partition in the subtree with the root p there exists a partition in the same place in the subtree with the root c and such that they are complementary in $\alpha = \beta$.

Let $\alpha_i = \beta_i$ be an attribute-value pair that is applicable to partitions that contain $\alpha = \beta$ but not applicable to partitions that contain $\alpha = \neg\beta$. That means that $\alpha_i = \beta_i$ can only appear in trees that contain $\alpha = \beta$, which means that having indicator for β set to true enables creating other child nodes of α and thus allows for $\alpha_i = \beta_i$ to be applied. Then, by Def. 3,

$\alpha = \beta$ is among the superiors for $\alpha_i = \beta_i$. This is in contradiction with the assumption that $\alpha = \beta$ is not among the superiors for any attribute-value pair $\alpha_i = \beta_i$, $i = 1, \ldots, k$. □

Statement 7. *Let T be the tree of partitions created by applying ordered list of attribute-value pairs $\alpha_1 = \beta_1, \ldots, \alpha_k = \beta_k, \alpha = \beta, \alpha_{k+1} = \beta_{k+1}, \ldots, \alpha_n = \beta_n$ and $\alpha = \beta$ is not among superiors for any attribute-value pair $\alpha_i = \beta_i$, $i = 1, \ldots, k$. Then, $\alpha = \beta$ can be pruned from T.*

Proof. According to Stat. 6 there exists at least one pair of partitions p and c which are complementary in $\alpha = \beta$ and if they are roots of subtrees, then for every partition in one subtree there exists a partition in the same place in the other subtree such that they are complementary in $\alpha = \beta$. If for all such pairs p and c, the subtree with c as the root is removed and the indicator for β is set to true in each partition in the subtree with p as the root, the resulting tree does not have any partitions created by applying $\alpha = \beta$ or created from the partitions that had $\alpha = \beta$ applied to it. □

Statement 8. *Any attribute-value pair that is used for building a tree of partitions can be pruned from that tree.*

Proof. Let $\alpha = \beta$ be the attribute value pair that is to be pruned from tree of partitions T created by $\alpha_1 = \beta_1, \ldots, \alpha_k = \beta_k, \alpha = \beta, \alpha_{k+1} = \beta_{k+1}, \ldots, \alpha_n = \beta_n$. If $\alpha = \beta$ is not among any superiors of $\alpha_{k+1} = \beta_{k+1}, \ldots, \alpha_n = \beta_n$ then Stat. 7 applies.

Assume $\alpha = \beta$ is among the superiors for $\alpha_{k_i} = \beta_{k_i}$ from $\alpha_{k+1} = \beta_{k+1}, \ldots, \alpha_n = \beta_n$ and assume $\alpha_{k_i} = \beta_{k_i}$ is not among superiors for any of $\alpha_{k_{i+1}} = \beta_{k_{i+1}}, \ldots, \alpha_n = \beta_n$. Let partitions p and c be complementary in $\alpha = \beta$ and be the roots of subtrees T_p and T_c. Then, according to Stat. 7

$\alpha_i = \beta_i$ can be pruned from subtree T_c and the resulting subtree T'_c is the same one as the one created when $\alpha_{k+1} = \beta_{k+1}, \ldots, \alpha_{k_{i-1}} = \beta_{k_{i-1}}, \alpha_{k_{i+1}} = \beta_{k_{i+1}}, \ldots, \alpha_n = \beta_n$ are applied to partition c. Then, if $\alpha = \beta$ is not among superiors for any of $\alpha_{k+1} = \beta_{k+1}, \ldots, \alpha_{k_{i-1}} = \beta_{k_{i-1}}, \alpha_{k_{i+1}} = \beta_{k_{i+1}}, \ldots, s_n = \beta_n$, according to Stat. 7, it can be pruned. Otherwise the same applies to any such attribute-value pair from $\alpha_{k+1} = \beta_{k+1}, \ldots, \alpha_{k_{i-1}} = \beta_{k_{i-1}}, \alpha_{k_{i+1}} = \beta_{k_{i+1}}, \ldots, \alpha_n = \beta_n$, $\alpha = \beta$ is the superior of.

If $\alpha_{k_i} = \beta_{k_i}$ is among superiors for any of $\alpha_{k_{i+1}} = \beta_{k_{i+1}}, \ldots, \alpha_n = \beta_n$ the same procedure is applied recursively until the attribute-value pair that is not among superiors for any of the attribute-value pairs following in the list is found and pruned. The last attribute-value pair trivially satisfies that requirement.

□

B.2 Pseudo code for pruning

```
void Part::prune(AttributeValuePair d)
{
  PartPtr upper, lower;
  Stack<float>& s;
  if(this->has_children)
    for each child c of this starting from the oldest
      c->prune(d);
  if (this->contains(complement(d)))
  {
    upper=this;
    for each child c of upper starting from the oldest
      if(c->contains(d))
      {
        lower=c;
        break;
      }
    lower->deletePartitions(s);
    s.push(lower->belief);
    delete(lower);
    upper->updateBeliefandRemoveComplements(s,d);
  }
}
void Part::deletePartitions(Stack<float>& s)
{
  for each child c of this starting from the oldest
  {
    c->deletePartitions(s);
    c->push(belief(c));
  }
  delete(children);
}
void Part::updateBeliefandRemoveComplements(Stack<float>&d, AttributeValuePair d)
{
  this->belief+=s.pop();
  this->removeComplement(d);
  for each child c starting from oldest that contains negation of d
  {
    c->updateBeliefandRemoveNegations(s,d);
    if(s->size()==0)
     break;
  }
}
```

Appendix C

CamInfo domain

C.1 CamInfo ontology

The CamInfo database has a tourist information for Cambridge, so that the user can ask for information about a restaurant, a bar, a hotel, a museum or other tourist attractions in the local area. The database consists of approximately 500 entities, each of which has up to 10 attributes that the user can query. The possible attribute-value pairs are organised in an hierarchical ontology, see Table C.1.

entity	←	*venue*(**type**, name, area, near, addr, phone, postcode)
type	←	*placetostay*(**staytype**, hasinternet, hasparking, price, pricerange, stars)
type	←	*placetoeat*(**eattype**, pricerange, openhours, price)
type	←	*placetodrink*(**drinktype**, pricerange, openhours, price)
type	←	*placetosee*(**seetype**, pricerange, openhours)
type	←	*entsvenue*(**entstype**)
type	←	*univenue*(**unitype**, openhours)
type	←	*sportsvenue*(**sport**)
type	←	*transvenue*(**transtype**)
type	←	*shopvenue*(**shoptype**, openhours)

Table C.1: CamInfo ontology rules.

type	$\leftarrow$	*amenity*(**amtype**)
amtype	$\leftarrow$	*hospital*()
amtype	$\leftarrow$	*policestation*()
amtype	$\leftarrow$	*bank*(openhours)
amtype	$\leftarrow$	*postoffice*(openhours)
amtype	$\leftarrow$	*touristinfo*(openhours)
shoptype	$\leftarrow$	*supermarket*()
shoptype	$\leftarrow$	*shoppingcentre*()
transtype	$\leftarrow$	*airport*()
transtype	$\leftarrow$	*busstation*()
transtype	$\leftarrow$	*trainstation*(openhours)
staytype	$\leftarrow$	*guesthouse*()
staytype	$\leftarrow$	*hotel*()
eattype	$\leftarrow$	*restaurant*(food)
drinktype	$\leftarrow$	*bar*(childrenallowed, hasinternet, hasmusic, hastv, openhours, price)
drinktype	$\leftarrow$	*coffeeshop*()
drinktype	$\leftarrow$	*pub*(childrenallowed, hasfood, hasinternet, hastv)
seetype	$\leftarrow$	*architecture*()
seetype	$\leftarrow$	*museum*()
seetype	$\leftarrow$	*park*()
unitype	$\leftarrow$	*college*()
unitype	$\leftarrow$	*department*()
unitype	$\leftarrow$	*library*()
entstype	$\leftarrow$	*cinema*()
entstype	$\leftarrow$	*theatre*()
entstype	$\leftarrow$	*nightclub*(openhours, price, pricerange)
entstype	$\leftarrow$	*entertainment*()

Table C.1: CamInfo ontology rules.

entstype	$\leftarrow$	*boat*()
entstype	$\leftarrow$	*concerthall*()
food	=	{ *American*, *Cafe food*, *Chinese*, ... }
pricerange	=	{ *free*, *cheap*, *moderate* , ... }
sport	=	{ *badmintoncourt*, *cricketfield*, *footballfield* , ... }
area	=	{ *girton*, *kingshedges*, *arbury* , ... }
...		

Table C.1: CamInfo ontology rules.

C.2 Typical long CamInfo dialogue requiring negotiation

In Table C.2 an example of a typical long dialogue requiring negotiation is given. It consists of 17 dialogue turns in each of which the system prompt and the true user action are given. In addition, the system's dialogue act and a scored list of the N-best user dialogue acts are provided, together with the probability of the most likely hypothesis and a sorted list of applied attribute-value pairs with their marginal probabilities. The attribute-value pairs that are in bold font match the most likely partition.

There are several points of the dialogue to note. Firstly, turns 4-6 show how the user goal changes during the dialogue. Secondly, turns 8 and 11 show the capability of the system to deal with long N-best inputs[1]. Furthermore, there are examples of how the system deals with a range of problems, such as mis-recognitions in turns 9, 11 and 12, a user mistake in turn 10 and a silence detection problem in turn 13. Moreover, in turns 8, 9 and 11-16 pruning of the lowest probability attribute value pairs takes place. Finally, it is interesting to note how the probability of the top hypothesis changes over time. For example, in turn 12, where a mis-recognition occurs, the probability of the top hypothesis is very low, only 0.013. This is due

[1]The confidence scores are here rounded to two decimals, but in fact they are always greater than zero.

to the fact that the user through-out the dialogue asked for a place to stay and in turn 12 the top partition contains a restaurant. However, in turn 14 and 15 when the system offers a hotel and user asks for an address, the probability increases from 0.025 to 0.308 showing how the system became more confident about the user's requirements.

1	System act	hello()
	System prompt	Thank you for calling the Cambridge Information system. How may I help you?
	True user action	HI I'M LOOKING FOR A PLACE TO STAY
	N-best input	hello(type=placetostay,task=find) 1.00
		hello(type=placetostay) 0.00
		hello(type=placetostay,name="The Place",task=find) 0.00
	Top hypothesis prb	0.676
	Applied attributes	**task=find** 0.997; **entity=venue** 0.994; **type=placetostay** 0.987; name=The Place 0.312;
2	System act	confreq(type=placetostay,near)
	System prompt	You are looking for a place to stay. Are you looking for somewhere near a particular venue?
	True user action	MAYBE SOMEWHERE IN THE CENTRE
	N-best input	inform(area=citycentre) 1.00
	Top hypothesis prb	0.900
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **area=citycentre** 0.972; **type=placetostay** 0.932; name=The Place 0.012;
3	System act	inform(name="Gonville Hotel", type=placetostay, area=citycentre)
	System prompt	The Gonville Hotel is a great place to stay and it's in the central part of town.
	True user action	IS IT EXPENSIVE

Table C.2: A typical long negotiative dialogue.

	N-best input	confirm(pricerange=expensive) 1.00 inform(pricerange=expensive) 0.00
	Top hypothesis prb	0.456
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **type=placetostay** 0.991; **name=Gonville Hotel** 0.978; **area=citycentre** 0.930; **pricerange=expensive** 0.495; name=The Place 0.006;
4	System act	inform(name="Gonville Hotel",pricerange=none)
	System prompt	I am sorry but there is no information about the price range for the Gonville Hotel.
	True user action	DO YOU HAVE ANY CHEAP HOTELS
	N-best input	inform(pricerange=cheap,=hotel) 0.96 inform(=hotel) 0.04
	Top hypothesis prb	0.290
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **type=placetostay** 0.987; **area=citycentre** 0.806; **staytype=hotel** 0.708; **pricerange=cheap** 0.585; name=Gonville Hotel 0.523; pricerange=expensive 0.201; name=The Place 0.014;
5	System act	inform(name=none, type=placetostay, area=citycentre, pricerange!=dontcare)
	System prompt	There is no place to stay in the central area for which I have any information about the price range.
	True user action	HOW ABOUT RIVERSIDE
	N-best input	reqalts(area=riverside) 0.90 inform(area=riverside) 0.05 reqalts() 0.04

Table C.2: A typical long negotiative dialogue.

		reqalts(=bar,area=riverside) 0.01
		null() 0.00
	Top hypothesis prb	0.213
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **area=riverside** 0.775; **type=placetostay** 0.752; **staytype=hotel** 0.585; **pricerange=cheap** 0.419; name=Gonville Hotel 0.273; type=placetodrink 0.198; area=citycentre 0.152; pricerange=expensive 0.150; drinktype=bar 0.092; name=The Place 0.068;
6	System act	inform(name=none,type=placetostay,pricerange=cheap)
	System prompt	There is no place to stay in the cheap price range.
	True user action	HOW ABOUT SOMETHING MODERATE
	N-best input	reqalts(pricerange=moderate) 0.83
		inform(pricerange=moderate) 0.17
	Top hypothesis prb	0.065
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **pricerange=moderate** 0.847; **area=riverside** 0.612; **type=placetostay** 0.593; type=placetodrink 0.394; staytype=hotel 0.338; name=Gonville Hotel 0.247; area=citycentre 0.218; drinktype=bar 0.193; name=The Place 0.138; pricerange=cheap 0.037; pricerange=dontcare 0.036; pricerange=expensive 0.033;
7	System act	inform(name=none, type=placetostay, staytype=hotel, area=riverside, pricerange=moderate)
	System prompt	There is no hotel in the riverside area in the moderate price range.
	True user action	I DON'T CARE ABOUT THE AREA

Table C.2: A typical long negotiative dialogue.

	N-best input	inform(=dontcare) 1.00 null() 0.00
	Top hypothesis prb	0.116

	Applied attributes	**task=find** 0.999; **entity=venue** 0.998;
		name=dontcare 0.569; type=placetostay 0.512;
		area=dontcare 0.491; **type=placetodrink** 0.458;
		pricerange=dontcare 0.433;
		pricerange=moderate 0.357; **drinktype=bar** 0.252;
		area=riverside 0.252; staytype=hotel 0.249;
		area=citycentre 0.131; name=Gonville Hotel 0.124;
		name=The Place 0.103; pricerange=cheap 0.061;
		pricerange=expensive 0.059;

8	System act	inform(name="The Eagle", type=placetodrink, drinktype=bar, area=citycentre, pricerange=none)
	System prompt	The Eagle, which is in the central part of town, is a great bar. I am sorry but there is no information about the price range for it.
	True user action	I'M LOOKING FOR A PLACE TO STAY
	N-best input	inform(type=placetostay,task=find) 0.99 inform(type=placetostay, name="The Place", task=find) 0.01 inform(type=placetostay) 0.00 inform(=trainstation, name="The Place", task=find) 0.00 inform(type=placetostay, name="Cambridge Police Station", task=find) 0.00 hello(type=placetostay) 0.00

Table C.2: A typical long negotiative dialogue.

		3 attribute-value pairs are pruned
	Top hypothesis prb	0.080
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **type=placetostay** 0.814; **pricerange=moderate** 0.516; **staytype=hotel** 0.421; **area=riverside** 0.321; pricerange=dontcare 0.259; area=dontcare 0.254; area=citycentre 0.196; type=placetodrink 0.125; name=Gonville Hotel 0.093; drinktype=bar 0.065; **name=dontcare** 0.065; name=The Eagle 0.049; name=Cambridge Police Station 0.049; name=The Place 0.049; type=transvenue 0.039; transtype=trainstation 0.020;
9	System act	inform(name=none, type=placetostay, staytype=hotel, area=riverside, pricerange=moderate)
	System prompt	There is no hotel in the riverside area in the moderate price range.
	True user action	I DON'T CARE ABOUT THE AREA
	N-best input	inform(=dontcare) 0.57
		null() 0.28
		inform() 0.15
		7 attribute-value pairs are pruned
	Top hypothesis prb	0.069

Table C.2: A typical long negotiative dialogue.

	Applied attributes	**task=find** 0.998; **entity=venue** 0.995; **type=placetostay** 0.958; **pricerange=dontcare** 0.435; staytype=hotel 0.420; **area=dontcare** 0.386; pricerange=moderate 0.365; **name=dontcare** 0.205; area=riverside 0.187; area=citycentre 0.182; name=Gonville Hotel 0.117;
10	System act	inform(name="City Centre North", type=placetostay, staytype=guesthouse, area=arbury, pricerange=none)
	System prompt	City Centre North is an outstanding guesthouse in the Arbury part of town, also I am sorry but there is no information about the price range for it.
	True user action	DO YOU HAVE ANY MODERATE RESTAURANT UH HOTEL
	N-best input	inform(pricerange=moderate, =restaurant, =hotel) 0.91 inform(=restaurant, =hotel) 0.05 inform(pricerange=moderate, =restaurant) 0.04
	Top hypothesis prb	0.023

Table C.2: A typical long negotiative dialogue.

	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **type=placetostay** 0.743; **pricerange=dontcare** 0.399; pricerange=moderate 0.362; name=City Centre North 0.301; **area=dontcare** 0.277; staytype=guesthouse 0.261; type=placetoeat 0.216; staytype=hotel 0.209; **name=dontcare** 0.188; area=citycentre 0.182; area=riverside 0.169; area=arbury 0.169; eattype=restaurant 0.124; name=Gonville Hotel 0.101;
11	System act	inform(name="Finches Bed and Breakfast", type=placetostay, area=girton, pricerange=none, staytype=guesthouse)
	System prompt	Finches Bed and Breakfast is a great guesthouse in the Girton part of town and I am sorry but there is no information about the price range for it.
	True user action	I'M LOOKING FOR A MODERATELY PRICED PLACE TO STAY
	N-best input	inform(pricerange=moderate, task=find) 0.79 inform(name="The Place", pricerange=moderate, task=find) 0.09 inform(type=placetostay, pricerange=moderate, task=find) 0.06 inform(pricerange=free, pricerange=moderate, task=find) 0.04 inform(type=placetostay, pricerange=moderate, name="The Place", task=find) 0.01 **3 attribute-value pairs are pruned**

Table C.2: A typical long negotiative dialogue.

	Top hypothesis prb	0.062
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998 **type=placetostay** 0.974; staytype=guesthouse 0.344; staytype=hotel 0.270; **area=dontcare** 0.206; area=girton 0.168; area=riverside 0.155; area=arbury 0.150; area=citycentre 0.134; name=City Centre North 0.063; pricerange=dontcare 0.074; pricerange=free 0.072; name=dontcare 0.064; name=Finches Bed and Breakfast 0.063; name=The Place 0.063;
12	System act	inform(name="Express by Holiday Inn", area=barnwell, type=placetostay, hasinternet=true, hasparking=true, stars="2")
	System prompt	The Express Holiday Inn has an internet connection and a parking lot and it's in the Barnwell part of town. This place, which has 2 stars, is a great place to stay.
	True user action	WHAT'S THE PRICERANGE
	N-best input	inform(=restaurant) 0.89
		null() 0.11
		5 attribute-value pairs are pruned
	Top hypothesis prb	0.013

Table C.2: A typical long negotiative dialogue.

	Applied attributes	**task=find** 0.999; **entity=venue** 0.999; type=placetostay 0.815; pricerange=moderate 0.594; hasinternet=true 0.408; hasparking=true 0.408; stars=2 0.407; name=Express by Holiday Inn 0.327; staytype=guesthouse 0.279; staytype=hotel 0.251; name=City Centre North 0.224; **type=placetoeat** 0.17; **area=dontcare** 0.161; **eattype=restaurant** 0.146; area=barnwell 0.145; area=riverside 0.142; area=girton 0.141; area=arbury 0.133; area=citycentre 0.129;
13	System act	inform(name="Riverside Restaurant and Granta Bar", area=riverside, eattype=restaurant, type=placetoeat, food=Traditional, pricerange=expensive, near="Queens' College")
	System prompt	The Riverside Restaurant is an outstanding restaurant in the riverside part of town in the expensive price range near Queens' College, also it serves traditional food.
	True user action	I'M LOOKING
	N-best input	inform(task=find) 0.50 null() 0.50 **8 attribute-value pairs are pruned**
	Top hypothesis prb	0.008

Table C.2: A typical long negotiative dialogue.

	Applied attributes	**task=find** 0.999; **entity=venue** 0.997; type=placetostay 0.830; hasinternet=true 0.422; hasparking=true 0.422; stars=2 0.416; pricerange=moderate 0.414; staytype=guesthouse 0.408; pricerange=expensive 0.286; area=dontcare 0.260; **area=riverside** 0.252; area=barnwell 0.243; name=Express by Holiday Inn 0.170; name=Riverside Restaurant and Granta Bar 0.168; type=placetoeat 0.146; eattype=restaurant 0.095; food=Traditional 0.047;
14	System act	inform(name="The Cricketers", area=riverside, near="Ruskin Gallery")
	System prompt	The Cricketers is in the riverside part of town near the Ruskin Gallery.
	True user action	I'M LOOKING FOR A FOR A HOTEL IN THE MODERATE PRICERANGE
	N-best input	inform(pricerange=moderate, =hotel, task=find) 0.96 inform(=hotel, pricerange=moderate) 0.04 **5 attribute-value pairs are pruned**
	Top hypothesis prb	0.025

Table C.2: A typical long negotiative dialogue.

	Applied attributes	**task=find** 0.999; **entity=venue** 0.999; **type=placetostay** 0.993; **pricerange=moderate** 0.722; **staytype=hotel** 0.722; hasinternet=true 0.503; hasparking=true 0.503; stars=2 0.493; textbfarea=dontcare 0.264; area=riverside 0.244; area=barnwell 0.244; name=The Cricketers 0.206; staytype=guesthouse 0.135; pricerange=expensive 0.135;
15	System act	inform(name="The Cambridge Belfry", type=placetostay, staytype=hotel, pricerange=moderate, area=girton, hasinternet=true, hasparking=true)
	System prompt	The Cambridge Belfry, which has an internet connection and parking spaces, is an outstanding hotel. It's in the moderate price range and in the Girton part of town.
	True user action	WHAT'S THE ADDRESS
	N-best input	request(addr) 1.00
		3 attribute-value pairs are pruned
	Top hypothesis prb	0.308

Table C.2: A typical long negotiative dialogue.

	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **type=placetostay** 0.990; **pricerange=moderate** 0.854; **staytype=hotel** 0.854; **name=The Cambridge Belfry** 0.794; **hasinternet=true** 0.791; **hasparking=true** 0.791; **area=dontcare** 0.390; **area=girton** 0.365; stars=2 0.199; area=barnwell 0.081; area=riverside 0.081;
16	System act	inform(name="The Cambridge Belfry", addr="Back Lane, Cambourne")
	System prompt	The Cambridge Belfry is located on Back Lane, Cambourne.
	True user action	AND THE PHONE NUMBER
	N-best input	request(phone) 1.00
		2 attribute-values are pruned
	Top hypothesis prb	0.427
	Applied attributes	**task=find** 0.999; **entity=venue** 0.998; **type=placetostay** 0.993; **hasinternet=true** 0.912; **hasparking=true** 0.912; **pricerange=moderate** 0.912; stars=2 0.081; **staytype=hotel** 0.912; **name=The Cambridge Belfry** 0.914; **area=dontcare** 0.482; area=girton 0.460;
17	System act	inform(name="The Cambridge Belfry", phone="01954714600")
	System prompt	The phone number of The Cambridge Belfry is 01954714600.
	True user action	THANKS BYE

Table C.2: A typical long negotiative dialogue.

	Hanging up

Table C.2: A typical long negotiative dialogue.

Appendix D

Gaussian processes in reinforcement learning

D.1 GP-Sarsa for episodic tasks

For a set of visited belief state-action pairs $\mathbf{B}_t = [(\mathbf{b}^0, a^0), \ldots, (\mathbf{b}^t, a^t)]^\mathsf{T}$, observed immediate rewards $\mathbf{r}_t = [r^1, \ldots, r^t]^\mathsf{T}$ and the dictionary of m representative points $\mathcal{D}_t = \{(\tilde{\mathbf{b}}^0, \tilde{a}^0), \ldots, (\tilde{\mathbf{b}}^m, \tilde{a}^m)\}$, the posterior of the Q-function (repeated from Eqs. 6.19, 6.10, 6.9d, 6.9a) can be written as:

$$Q(\mathbf{b}, a)|\mathbf{B}_t, \mathbf{r}_t \sim \mathcal{N}(\tilde{\mathbf{k}}_t(\mathbf{b}, a)^\mathsf{T} \tilde{\boldsymbol{\mu}}_t, k((\mathbf{b}, a), (\mathbf{b}, a)) - \tilde{\mathbf{k}}_t(\mathbf{b}, a)^\mathsf{T} \tilde{\mathbf{C}}_t \tilde{\mathbf{k}}_t(\mathbf{b}, a)), \tag{D.1a}$$

$$\tilde{\boldsymbol{\mu}}_t = \tilde{\mathbf{H}}_t^\mathsf{T} \tilde{\mathbf{W}}_t \mathbf{r}_t, \tag{D.1b}$$

$$\tilde{\mathbf{C}}_t = \tilde{\mathbf{H}}_t^\mathsf{T} \tilde{\mathbf{W}}_t \tilde{\mathbf{H}}_t, \tag{D.1c}$$

$$\tilde{\mathbf{W}}_t = (\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\mathsf{T} + \sigma^2 \tilde{\mathbf{H}}_t \tilde{\mathbf{H}}_t^\mathsf{T})^{-1}, \tag{D.1d}$$

$$\tilde{\mathbf{H}}_t = \mathbf{H}_t \mathbf{G}_t, \tag{D.1e}$$

$$\mathbf{H}_t = \begin{bmatrix} 1 & -\gamma & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 & -\gamma \end{bmatrix}, \tag{D.1f}$$

$$\mathbf{G}_t = [(\tilde{\mathbf{K}}_t^{-1}\tilde{\mathbf{k}}_t(\mathbf{b}^0, a^0))^\mathsf{T}, \ldots, (\tilde{\mathbf{K}}_t^{-1}\tilde{\mathbf{k}}_t(\mathbf{b}^t, a^t))^\mathsf{T}]^\mathsf{T}, \tag{D.1g}$$

and

$$\tilde{\mathbf{k}}_t(\mathbf{b}, a) = [k((\mathbf{b}, a), (\tilde{\mathbf{b}}^0, \tilde{a}^0)), \ldots, k((\mathbf{b}, a), (\tilde{\mathbf{b}}^m, \tilde{a}^m))]^\mathsf{T}, \tag{D.1h}$$

$$\tilde{\mathbf{K}} = [\tilde{\mathbf{k}}_t(\tilde{\mathbf{b}}^0, \tilde{a}^0)^\mathsf{T}, \ldots, \tilde{\mathbf{k}}_t(\tilde{\mathbf{b}}^m, \tilde{a}^m)^\mathsf{T}]^\mathsf{T}. \tag{D.1i}$$

For two consequent belief state-action pairs and an associated reward ($\mathbf{b}^t$, a^t, r^{t+1}, $\mathbf{b}^{t+1}$, a^{t+1}), the posterior $Q(\mathbf{b}, a)|\mathbf{B}_{t+1}, \mathbf{r}_{t+1}$ needs to be calculated efficiently, where $\mathbf{B}_{t+1} = [\mathbf{B}_t^\mathsf{T}, (\mathbf{b}^{t+1}, a^{t+1})]^\mathsf{T}$ and $\mathbf{r}_{t+1} = [\mathbf{r}_t^\mathsf{T}, r^{t+1}]^\mathsf{T}$. In [125] the sufficient statistics that enable this with complexity $O(m^2)$, where m is the size of the dictionary, are given for non-episodic tasks. Here, this is extended to support episodic tasks.

The main difference between episodic and non-episodic tasks is in the way the Q-function and the reward are related.

In the case where $\mathbf{b}^t$ is not an initial turn and $\mathbf{b}^{t+1}$ is not the final turn in the episode, the relation follows from Eq. 6.8:

$$\begin{aligned} r^t &= Q^\pi(\mathbf{b}^{t-1}, a^{t-1}) - \gamma Q^\pi(\mathbf{b}^t, a^t) + \Delta Q^\pi(\mathbf{b}^{t-1}, a^{t-1}) - \gamma \Delta Q^\pi(\mathbf{b}^t, a^t) \\ r^{t+1} &= Q^\pi(\mathbf{b}^t, a^t) - \gamma Q^\pi(\mathbf{b}^{t+1}, a^{t+1}) + \Delta Q^\pi(\mathbf{b}^t, a^t) - \gamma \Delta Q^\pi(\mathbf{b}^{t+1}, a^{t+1}). \end{aligned} \tag{D.2}$$

In the case where $\mathbf{b}^{t+1}$ is the final belief state in the episode, there is no Q-function or residual associated with it, so the reward r^{t+1} relates to the Q-function and its residual in belief state $\mathbf{b}^t$:

$$\begin{aligned} r^t &= Q^\pi(\mathbf{b}^{t-1}, a^{t-1}) - \gamma Q^\pi(\mathbf{b}^t, a^t) + \Delta Q^\pi(\mathbf{b}^{t-1}, a^{t-1}) - \gamma \Delta Q^\pi(\mathbf{b}^t, a^t) \\ r^{t+1} &= Q^\pi(\mathbf{b}^t, a^t) + \Delta Q^\pi(\mathbf{b}^t, a^t). \end{aligned} \tag{D.3}$$

In the case where $\mathbf{b}^t$ is an initial turn of episode e, $t = 0$, both belief state $\mathbf{b}^0$ and belief state $\mathbf{b}^1$ are visited for the first time. The previously obtained reward $r^{T_{e-1}+1}$ is only associated with the last (non-terminal) belief state of the previous episode $e-1$. The relation between the Q-function and the reward in this case is:

$$\begin{aligned} r^{T_{e-1}+1} &= Q^\pi(\mathbf{b}^{T_{e-1}}, a^{T_{e-1}}) + \Delta Q^\pi(\mathbf{b}^{T_{e-1}}, a^{T_{e-1}}) \\ r^1 &= Q^\pi(\mathbf{b}^0, a^0) - \gamma Q^\pi(\mathbf{b}^1, a^1) + \Delta Q^\pi(\mathbf{b}^0, a^0) - \gamma \Delta Q^\pi(\mathbf{b}^1, a^1). \end{aligned} \quad (D.4)$$

For each of these three cases a description of sufficient statistics is given in the following sections, which enables the posterior at turn $t+1$ to be calculated in $O(m^2)$ operations, where m the size of the dictionary.

D.1.1 Case 1: $\mathbf{b}^t$ non-initial belief state and $\mathbf{b}^{t+1}$ non-terminal belief state

The case where $\mathbf{b}^t$ is a non-initial belief state and $\mathbf{b}^{t+1}$ is a non-terminal belief state coincides with the description given in [125]. For clarity, the main points are reproduced here.

From Eq. D.2 the relation between the previous $\mathbf{H}_t$ (defined in Eq. D.1f) and new $\mathbf{H}_{t+1}$ is given by:

$$\mathbf{H}_{t+1} = \begin{bmatrix} \mathbf{H}_t & & \mathbf{0} \\ \mathbf{0}^\top & 1 & -\gamma \end{bmatrix}. \quad (D.5)$$

Since $\mathbf{b}^t$ is not the initial belief state, it has been visited at the previous turn. However, belief state $\mathbf{b}^{t+1}$ is visited for the first time. If the sparsification threshold is exceeded belief state $\mathbf{b}^{t+1}$ must be included in the dictionary.

First is the case when the dictionary remains the same, $\mathcal{D}_{t+1} = \mathcal{D}_t$. Then the approximation of the Gram matrix also remains the same: $\tilde{\mathbf{K}}_{t+1} = \tilde{\mathbf{K}}_t$. Since the sparsification threshold is not exceeded, the vector of coefficients for belief state $\mathbf{b}^{t+1}$ is $\mathbf{g}_{t+1} = \tilde{\mathbf{K}}_t^{-1}\tilde{\mathbf{k}}_t(\mathbf{b}^{t+1})$:

$$\mathbf{G}_{t+1} = \begin{bmatrix} \mathbf{G}_t \\ \mathbf{g}_{t+1}^\top \end{bmatrix}. \quad (D.6)$$

From Eq. D.5 and Eq. D.1a:

$$\tilde{\mathbf{H}}_{t+1} = \begin{bmatrix} \tilde{\mathbf{H}}_t \\ \tilde{\mathbf{h}}_{t+1}^\top \end{bmatrix}, \tag{D.7}$$

where $\tilde{\mathbf{h}}_{t+1} = \mathbf{g}_{t+1} - \gamma \mathbf{g}_t$. This then allows $\tilde{\mathbf{W}}_{t+1}^{-1}$ (defined in Eq. D.1a) to be calculated using $\tilde{\mathbf{W}}_t^{-1}$:

$$\begin{aligned} \tilde{\mathbf{W}}_{t+1}^{-1} &= \begin{bmatrix} \tilde{\mathbf{H}}_t\tilde{\mathbf{K}}_t\tilde{\mathbf{H}}_t^\top + \tilde{\mathbf{H}}_t\tilde{\mathbf{H}}_t^\top & \tilde{\mathbf{H}}_t\tilde{\mathbf{K}}_t\tilde{\mathbf{h}}_{t+1} + \tilde{\mathbf{H}}_t\tilde{\mathbf{h}}_{t+1} \\ \tilde{\mathbf{h}}_{t+1}^\top\tilde{\mathbf{K}}_t\tilde{\mathbf{H}}_t + \tilde{\mathbf{h}}_{t+1}^\top\tilde{\mathbf{H}}_t & \tilde{\mathbf{h}}_{t+1}^\top\tilde{\mathbf{H}}_t\tilde{\mathbf{h}}_{t+1} \end{bmatrix} \\ &= \begin{bmatrix} \tilde{\mathbf{W}}_t^{-1} & \tilde{\mathbf{H}}_t\Delta\tilde{\mathbf{k}}_{t+1} - \gamma\sigma^2\mathbf{u} \\ (\tilde{\mathbf{H}}_t\Delta\tilde{\mathbf{k}}_{t+1} - \gamma\sigma^2\mathbf{u})^\top & \tilde{\mathbf{h}}_{t+1}^\top\Delta\tilde{\mathbf{k}}_{t+1} + (1+\gamma^2)\sigma^2 \end{bmatrix}, \end{aligned} \tag{D.8}$$

where $\Delta\tilde{\mathbf{k}}_{t+1} = \tilde{\mathbf{K}}_t\tilde{\mathbf{h}}_{t+1} = \tilde{\mathbf{k}}(\mathbf{b}^t, a^t) - \gamma\tilde{\mathbf{k}}(\mathbf{b}^{t+1}, a^{t+1})$ and $\mathbf{u} = [\mathbf{0}^\top, 1]^\top$. Then, the recursive expression for $\tilde{\mathbf{W}}_{t+1}$ follows using the partitioned matrix inverse theorem [125, 146]:

$$\tilde{\mathbf{W}}_{t+1} = \frac{1}{v_{t+1}} \begin{bmatrix} \tilde{\mathbf{W}}_t + \mathbf{l}_t\mathbf{l}_t^\top & -\mathbf{l}_t^\top \\ -\mathbf{l}_t & 1 \end{bmatrix}, \tag{D.9}$$

where $\mathbf{l}_t = \tilde{\mathbf{W}}_t(\tilde{\mathbf{H}}_t\Delta\tilde{\mathbf{k}}_{t+1} - \gamma\sigma^2\mathbf{u})$ and $v_{t+1} = (1+\gamma^2)\sigma^2 + \tilde{\mathbf{h}}_{t+1}^\top\Delta\tilde{\mathbf{k}}_{t+1} + (\tilde{\mathbf{H}}_t\Delta\tilde{\mathbf{k}}_{t+1} - \gamma\sigma^2\mathbf{u})^\top\tilde{\mathbf{W}}_t^T(\tilde{\mathbf{H}}_t\Delta\tilde{\mathbf{k}}_{t+1} - \gamma\sigma^2\mathbf{u})$.

Replacing the expression for $\tilde{\mathbf{W}}_{t+1}$ from Eq. D.9 in expressions for $\tilde{\mathbf{C}}_{t+1}$ and $\tilde{\boldsymbol{\mu}}_{t+1}$ in Eq. D.1a allows the following recursive relations to be established:

$$\begin{aligned} \tilde{\boldsymbol{\mu}}_{t+1} &= \tilde{\boldsymbol{\mu}}_t + \tfrac{\tilde{\mathbf{c}}_{t+1}}{v_{t+1}} d_{t+1}, \\ \tilde{\mathbf{C}}_{t+1} &= \tilde{\mathbf{C}}_t + \tfrac{1}{v_{t+1}}\tilde{\mathbf{c}}_{t+1}\tilde{\mathbf{c}}_{t+1}^\top \\ v_{t+1} &= (1+\gamma^2)\sigma^2 + \Delta\tilde{\mathbf{k}}_{t+1}^\top(\tilde{\mathbf{c}}_{t+1} + \tfrac{\gamma\sigma^2}{v_t}\tilde{\mathbf{c}}_t) - \tfrac{\gamma^2\sigma^4}{v_t} \\ d_{t+1} &= \tfrac{\gamma\sigma^2}{v_t} d_t + r^{t+1} - \Delta\tilde{\mathbf{k}}_{t+1}^\top\tilde{\boldsymbol{\mu}}_t \\ \tilde{\mathbf{c}}_{t+1} &= \tfrac{\gamma\sigma^2}{v_t}\tilde{\mathbf{c}}_t + \tilde{\mathbf{h}}_{t+1} - \tilde{\mathbf{C}}_t\Delta\tilde{\mathbf{k}}_{t+1}. \end{aligned} \tag{D.10}$$

The expressions in Eq. D.10 represent the sufficient statistics for the posterior of the Q-function. The complexity of calculating the posterior is $O(m^2)$, where $m = |\mathcal{D}_{t+1}|$.

Second is the case when the next belief state $\mathbf{b}^{t+1}$ must be included in the dictionary, $\mathcal{D}_{t+1} = \{(\mathbf{b}^{t+1}, a^{t+1})\} \cup \mathcal{D}_t$. In that case the approximated Gram matrix $\tilde{\mathbf{K}}_{t+1}$ and the matrix of coefficients $\mathbf{G}_{t+1}$ are expanded:

$$\begin{aligned}
\tilde{\mathbf{K}}_{t+1} &= \begin{bmatrix} \tilde{\mathbf{K}}_t & \tilde{\mathbf{k}}_t(\mathbf{b}^{t+1}, a^{t+1}) \\ \tilde{\mathbf{k}}_t(\mathbf{b}^{t+1}, a^{t+1})^\intercal & k((\mathbf{b}^{t+1}, a^{t+1}), (\mathbf{b}^{t+1}, a^{t+1})) \end{bmatrix}, \\
\mathbf{G}_{t+1} &= \begin{bmatrix} \mathbf{G}_t & \mathbf{0} \\ \mathbf{0}^\intercal & 1 \end{bmatrix}.
\end{aligned} \tag{D.11}$$

Then from Eq. D.11 and Eq. D.1a:

$$\tilde{\mathbf{H}}_{t+1} = \begin{bmatrix} \tilde{\mathbf{H}}_t & \mathbf{0} \\ \tilde{\mathbf{h}}_{t+1}^\intercal & \end{bmatrix}, \tag{D.12}$$

where $\tilde{\mathbf{h}}_{t+1} = [\mathbf{g}_t^\intercal, -\gamma]^\intercal$ and $\mathbf{g}_t^\intercal$ is the last row of matrix $\mathbf{G}_t$, $\mathbf{g}_t = \tilde{\mathbf{K}}_t^{-1}\tilde{\mathbf{k}}(\mathbf{b}^t, a^t)$. It can be shown that then $\tilde{\mathbf{W}}_{t+1}^{-1}$ becomes:

$$\tilde{\mathbf{W}}_{t+1}^{-1} = \begin{bmatrix} \tilde{\mathbf{W}}_t^{-1} & \tilde{\mathbf{H}}_t \Delta\tilde{\mathbf{k}}_{t+1} \\ \Delta\tilde{\mathbf{k}}_{t+1}^\intercal \tilde{\mathbf{H}}_t^\intercal & \Delta k_{tt} + (1+\gamma^2)\sigma^2 \end{bmatrix}, \tag{D.13}$$

where $\Delta k_{tt} = \mathbf{g}_t^\intercal(\tilde{\mathbf{k}}(\mathbf{b}^t, a^t) - 2\gamma\tilde{\mathbf{k}}(\mathbf{b}^{t+1}, a^{t+1})) + \gamma^2 k((\mathbf{b}^{t+1}, a^{t+1}), (\mathbf{b}^{t+1}, a^{t+1}))$. In a similar way when the dictionary is not expanded, applying the partitioned matrix inverse theorem to expression for $\tilde{\mathbf{W}}_{t+1}^{-1}$ yields a recursive expression for $\tilde{\mathbf{W}}_{t+1}$. Replacing this in expressions for $\tilde{\boldsymbol{\mu}}_{t+1}$ and $\tilde{\mathbf{C}}_{t+1}$ gives:

$$\begin{aligned}
\tilde{\boldsymbol{\mu}}_{t+1} &= \begin{bmatrix} \tilde{\boldsymbol{\mu}}_t \\ 0 \end{bmatrix} + \tfrac{\tilde{\mathbf{c}}_{t+1}}{v_{t+1}} d_{t+1}, \\
\tilde{\mathbf{C}}_{t+1} &= \begin{bmatrix} \tilde{\mathbf{C}}_t & \mathbf{0} \\ \mathbf{0}^\intercal & 0 \end{bmatrix} + \tfrac{1}{v_{t+1}} \tilde{\mathbf{c}}_{t+1} \tilde{\mathbf{c}}_{t+1}^\intercal \\
v_{t+1} &= (1+\gamma^2)\sigma^2 + \Delta k_{tt} - \Delta\tilde{\mathbf{k}}_{t+1}^\intercal \tilde{\mathbf{C}}_t \Delta\tilde{\mathbf{k}}_{t+1} + \tfrac{2\gamma\sigma^2}{v_t} \tilde{\mathbf{c}}_t \Delta\tilde{\mathbf{k}}_{t+1} - \tfrac{\gamma^2\sigma^4}{v_t} \\
d_{t+1} &= \tfrac{\gamma\sigma^2}{v_t} d_t + r^{t+1} - \Delta\tilde{\mathbf{k}}_{t+1}^\intercal \tilde{\boldsymbol{\mu}}_t \\
\tilde{\mathbf{c}}_{t+1} &= \tfrac{\gamma\sigma^2}{v_t} \begin{bmatrix} \tilde{\mathbf{c}}_t \\ 0 \end{bmatrix} + \tilde{\mathbf{h}}_{t+1} - \begin{bmatrix} \tilde{\mathbf{C}}_t \Delta\tilde{\mathbf{k}}_{t+1} \\ 0 \end{bmatrix}.
\end{aligned} \tag{D.14}$$

The complexity of calculating the posterior remains $O(m^2)$ where $m = |\mathcal{D}_{t+1}|$.

D.1.2 Case 2: $\mathbf{b}^{t+1}$ terminal belief state

In the case where $\mathbf{b}^{t+1}$ is the terminal belief state, there is no action or Q-value associated with that belief state so it is not included in the dictionary. The reward only relates to the Q-value of belief state $\mathbf{b}^t$ and action a^t, as in Eq. D.3, so the relation between previous $\mathbf{H}_t$ and new $\mathbf{H}_{t+1}$ becomes:

$$\mathbf{H}_{t+1} = \begin{bmatrix} \mathbf{H}_t \\ \mathbf{0}^\mathsf{T} \quad 1 \end{bmatrix}. \tag{D.15}$$

The dictionary and the approximated Gram matrix remain the same, $\mathcal{D}_{t+1} = \mathcal{D}_t$ and $\tilde{\mathbf{K}}_{t+1} = \tilde{\mathbf{K}}_t$, and $\mathbf{G}_{t+1}$ and $\tilde{\mathbf{H}}_{t+1}$ become:

$$\mathbf{G}_{t+1} = \begin{bmatrix} \mathbf{G}_t \\ \mathbf{g}_t^\mathsf{T} \end{bmatrix}, \tilde{\mathbf{H}}_{t+1} = \begin{bmatrix} \tilde{\mathbf{H}}_t \\ \tilde{\mathbf{h}}_{t+1}^\mathsf{T} \end{bmatrix}, \tag{D.16}$$

where $\tilde{\mathbf{h}}_{t+1} = \mathbf{g}_t$ and $\mathbf{g}_t^\mathsf{T}$ the last row from $\mathbf{G}_t$ (associated with belief state $\mathbf{b}^t$). This allows for $\tilde{\mathbf{W}}_{t+1}^{-1}$ to be expressed using $\tilde{\mathbf{W}}_t^{-1}$:

$$\tilde{\mathbf{W}}_{t+1}^{-1} = \begin{bmatrix} \tilde{\mathbf{W}}_t^{-1} & \tilde{\mathbf{H}}_t \Delta\tilde{\mathbf{k}}_{t+1} - \gamma\sigma^2 \mathbf{u} \\ (\tilde{\mathbf{H}}_t \Delta\tilde{\mathbf{k}}_{t+1} - \gamma\sigma^2 \mathbf{u})^\mathsf{T} & \tilde{\mathbf{h}}_{t+1}^\mathsf{T} \Delta\tilde{\mathbf{k}}_{t+1} + \gamma^2\sigma^2 \end{bmatrix}, \tag{D.17}$$

where $\Delta\tilde{\mathbf{k}}_{t+1} = \tilde{\mathbf{k}}_t(\mathbf{b}^t, a^t)$. Applying the partitioned matrix inverse theorem to expression for $\tilde{\mathbf{W}}_{t+1}^{-1}$ yields $\tilde{\mathbf{W}}_{t+1}$. The recursive expressions for $\tilde{\boldsymbol{\mu}}_{t+1}$ and $\tilde{\mathbf{C}}_{t+1}$ then become:

$$\begin{aligned}
\tilde{\boldsymbol{\mu}}_{t+1} &= \tilde{\boldsymbol{\mu}}_t + \frac{\tilde{\mathbf{c}}_{t+1}}{v_{t+1}} d_{t+1}, \\
\tilde{\mathbf{C}}_{t+1} &= \tilde{\mathbf{C}}_t + \frac{1}{v_{t+1}} \tilde{\mathbf{c}}_{t+1} \tilde{\mathbf{c}}_{t+1}^\mathsf{T} \\
v_{t+1} &= \gamma^2\sigma^2 + \Delta\tilde{\mathbf{k}}_{t+1}^\mathsf{T} (\tilde{\mathbf{c}}_{t+1} + \frac{\gamma\sigma^2}{v_t} \tilde{\mathbf{c}}_t) - \frac{\gamma^2\sigma^4}{v_t} \\
d_{t+1} &= \frac{\gamma\sigma^2}{v_t} d_t + r^{t+1} - \Delta\tilde{\mathbf{k}}_{t+1}^\mathsf{T} \tilde{\boldsymbol{\mu}}_t \\
\tilde{\mathbf{c}}_{t+1} &= \frac{\gamma\sigma^2}{v_t} \tilde{\mathbf{c}}_t + \tilde{\mathbf{h}}_{t+1} - \tilde{\mathbf{C}}_t \Delta\tilde{\mathbf{k}}_{t+1},
\end{aligned} \tag{D.18}$$

which represent the sufficient statistics and the complexity is $O(m^2)$, where

$m = |\mathcal{D}_{t+1}|$.

D.1.3 Case 3: $\mathbf{b}^t$ initial belief state

In the case when $\mathbf{b}^t$ is the initial belief state of new episode e, the relation between previous $\mathbf{H}_{T_{e-1}}$ and new $\mathbf{H}_1$ follows from Eq. D.4:

$$\mathbf{H}_1 = \begin{bmatrix} \mathbf{H}_{T_{e-1}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0}^\intercal & 1 & -\gamma \end{bmatrix}. \tag{D.19}$$

In this case, both $\mathbf{b}^0$ and $\mathbf{b}^1$ are visited for the first time and both can potentially expand the dictionary. Therefore this case has four sub-cases.

First is the case when both $\mathbf{b}^0$ and $\mathbf{b}^1$ expand the dictionary, $\mathcal{D}_1 = \{(\mathbf{b}^t, a^t), (\mathbf{b}^{t+1}, a^{t+1})\} \cup \mathcal{D}_{T_{e-1}}$, where $\mathcal{D}_{T_{e-1}}$ is the dictionary from the previous episode. Then:

$$\begin{aligned} \tilde{\mathbf{G}}_1 &= \begin{bmatrix} \tilde{\mathbf{H}}_{T_{e_1}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0}^\intercal & 1 & 0 \\ \mathbf{0}^\intercal & 0 & 1 \end{bmatrix}, \\ \tilde{\mathbf{H}}_1 &= \begin{bmatrix} \tilde{\mathbf{H}}_{T_{e_1}} & \mathbf{0} & \mathbf{0} \\ & \tilde{\mathbf{h}}_1 & \end{bmatrix}, \\ \tilde{\mathbf{K}}_1 &= \begin{bmatrix} \tilde{\mathbf{K}}_{T_{e-1}} & \tilde{\mathbf{k}}_{T_{e-1}}(\mathbf{b}^0, a^0) & \tilde{\mathbf{k}}_{T_{e-1}}(\mathbf{b}^1, a^1) \\ \tilde{\mathbf{k}}_{T_{e-1}}(\mathbf{b}^0, a^0)^\intercal & k((\mathbf{b}^0, a^0), (\mathbf{b}^0, a^0)) & k((\mathbf{b}^0, a^0), (\mathbf{b}^1, a^1)) \\ \tilde{\mathbf{k}}_{T_{e-1}}(\mathbf{b}^1, a^1)^\intercal & k((\mathbf{b}^0, a^0), (\mathbf{b}^1, a^1)) & k((\mathbf{b}^1, a^1), (\mathbf{b}^1, a^1)) \end{bmatrix} \end{aligned} \tag{D.20}$$

where $\tilde{\mathbf{h}}_1 = [\mathbf{0}^\intercal, 1, -\gamma]^\intercal$. It can be then shown that $\tilde{\mathbf{W}}_1^{-1}$ becomes:

$$\tilde{\mathbf{W}}_1^{-1} = \begin{bmatrix} \tilde{\mathbf{W}}_{T_{e-1}}^{-1} & \mathbf{0} \\ \mathbf{0}^\intercal & (1+\gamma^2)\sigma^2 + \Delta k_{tt} \end{bmatrix}, \tag{D.21}$$

where $\Delta k_{tt} = k((\mathbf{b}^0, a^0), (\mathbf{b}^0, a^0)) - 2\gamma k((\mathbf{b}^0, a^0), (\mathbf{b}^1, a^1)) + \gamma^2 k((\mathbf{b}^1, a^1), (\mathbf{b}^1, a^1))$. $\tilde{\mathbf{W}}_1$ trivially follows from Eq. D.21 :

$$\tilde{\mathbf{W}}_1 = \begin{bmatrix} \tilde{\mathbf{W}}_{T_{e-1}} & \mathbf{0} \\ \mathbf{0}^\intercal & \frac{1}{v_1} \end{bmatrix}, \tag{D.22}$$

where $v_1 = (1+\gamma^2)\sigma + \Delta k_{tt}$, yielding the expressions for $\tilde{\boldsymbol{\mu}}_1$ and $\tilde{\mathbf{C}}_1$:

$$\begin{aligned}
\tilde{\boldsymbol{\mu}}_1 &= \begin{bmatrix} \tilde{\boldsymbol{\mu}}_{T_{e-1}} \\ 0 \\ 0 \end{bmatrix} + \tfrac{\tilde{\mathbf{c}}_1}{v_1} d_1, \\
\tilde{\mathbf{C}}_1 &= \begin{bmatrix} \tilde{\mathbf{C}}_{T_{e-1}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0 & 0 \\ \mathbf{0}^\mathsf{T} & 0 & 0 \end{bmatrix} + \tfrac{1}{v_1}\tilde{\mathbf{c}}_1 \tilde{\mathbf{c}}_1^\mathsf{T}, \\
v_1 &= (1+\gamma^2)\sigma^2 + \Delta k_{tt}, \\
d_1 &= r^1, \\
\tilde{\mathbf{c}}_1 &= \tilde{\mathbf{h}}_1.
\end{aligned} \tag{D.23}$$

Note that defining:

$$\begin{aligned}
\tilde{\mathbf{K}}_0 &= \begin{bmatrix} \tilde{\mathbf{K}}_{T_{e-1}} & \tilde{\mathbf{k}}_{T_{e-1}}(\mathbf{b}^1, a^1) \\ \tilde{\mathbf{k}}_{T_{e-1}}(\mathbf{b}^0, a^0)^\mathsf{T} & k((\mathbf{b}^0, a^0), (\mathbf{b}^0, a^0)) \end{bmatrix}, \\
\frac{1}{v_0} &= 0, \\
\boldsymbol{\mu}_0 &= \begin{bmatrix} \tilde{\boldsymbol{\mu}}_{T_{e-1}} \\ 0 \end{bmatrix}, \\
\tilde{\mathbf{C}}_0 &= \begin{bmatrix} \tilde{\mathbf{C}}_{T_{e-1}} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix},
\end{aligned} \tag{D.24}$$

and applying the same recursive expressions from Eq. D.14, yields the expressions in Eq D.23 since in this particular case it can be shown that $\tilde{\mathbf{C}}_0 \Delta \tilde{\mathbf{k}}_0(\mathbf{b}^{t+1}, a^{t+1}) = \mathbf{0}$.

Second is the case when neither $\mathbf{b}^0$ nor $\mathbf{b}^1$ expand the dictionary, so $\mathcal{D}_1 = \mathcal{D}_{T_{e-1}}$, $\tilde{\mathbf{K}}_1 = \tilde{\mathbf{K}}_{T_{e-1}}$.

$$\begin{aligned}
\mathbf{G}_1 &= \begin{bmatrix} \mathbf{G}_{T_{e-1}} \\ \mathbf{g}_0^\mathsf{T} \\ \mathbf{g}_1^\mathsf{T} \end{bmatrix}, \\
\tilde{\mathbf{H}}_1 &= \begin{bmatrix} \tilde{\mathbf{H}}_{T_{e_1}} \\ \tilde{\mathbf{h}}_1^\mathsf{T} \end{bmatrix},
\end{aligned} \tag{D.25}$$

where $\tilde{\mathbf{h}}_1 = \mathbf{g}_0 - \gamma \mathbf{g}_1$. When compared with Eq. D.7 it can bee seen that the structure is very similar. If the same procedure is applied as in Section D.1.1

it can be shown that the expressions for $\tilde{\boldsymbol{\mu}}_1$ and $\tilde{\mathbf{C}}_1$ are the same if the sufficient statistics at turn 0 were defined as

$$\begin{aligned} \tilde{\mathbf{K}}_0 &= \tilde{\mathbf{K}}_{T_{e-1}}, \\ \frac{1}{v_0} &= 0, \\ \boldsymbol{\mu}_0 &= \tilde{\boldsymbol{\mu}}_{T_{e-1}}, \\ \tilde{\mathbf{C}}_0 &= \tilde{\mathbf{C}}_{T_{e-1}}, \end{aligned} \tag{D.26}$$

and then Eq. D.10 applied. This is also the case when $\mathbf{b}^{t+1}$ needs to be included in the dictionary. In the case when $\mathbf{b}^t$ needs to be included in the dictionary but not $\mathbf{b}^{t+1}$, the sufficient statistics at turn 0 need to be defined as in Eq. D.24 and then Eq. D.10 can be applied. Therefore, the expressions at turn $t+1$ are the same regardless whether turn t is initial or not.

Algorithm 9 gives a full description of GP-Sarsa for episodic tasks. It recalculates the sufficient statistics of the Gaussian process model for the Q-function after every turn. The algorithm starts by initialising belief state $\mathbf{b}$. At the beginning of first episode action a is taken randomly, the dictionary is initialised and the inverse Gram matrix trivially calculated (line 5). Otherwise action a is taken ϵ-greedily with respect to the current estimate of $\tilde{\boldsymbol{\mu}}$ (line 7) which determines the mean of the Q-value (Eq. D.1a). The sufficient statistics are then copied from the end of the previous episode (line 9). If sparcification threshold is exceeded the dictionary is expanded and the sufficient statistics recalculated (lines 10-14).

For each step in the episode action a is taken, reward r' observed, belief state updated $\mathbf{b}'$ and next action a' chosen ϵ-greedily (line 16). In the final step of the episode, the system takes action a in the penultimate belief state $\mathbf{b}$, obtains the reward r' and then moves to the final belief state $\mathbf{b}'$. As explained, the next belief state $\mathbf{b}'$ and its residual are only taken into consideration when step is not final, lines 17-20 and lines 33-35 respectively. Also, only non-final belief states included in the dictionary (lines 24-30).

Algorithm 9 Episodic GP-Sarsa

```
1: Initialise $\tilde{\boldsymbol{\mu}} \leftarrow []$, $\tilde{\mathbf{C}} \leftarrow []$, $\tilde{\mathbf{c}} \leftarrow []$, $d \leftarrow 0$, $1/v \leftarrow 0$
2: for each episode do
3:   Initialise $\mathbf{b}$
4:   if first episode then
5:     choose $a$ arbitrary, $\mathcal{D} = \{(\mathbf{b}, a)\}$, $\tilde{\mathbf{K}}^{-1} \leftarrow 1/k((\mathbf{b}, a), (\mathbf{b}, a))$
6:   else
7:     choose action $a = \arg\max_a \tilde{\mathbf{k}}(\mathbf{b}, a)^\mathsf{T} \tilde{\boldsymbol{\mu}}$ or randomly with probability $\epsilon$
8:   end if
9:   $\tilde{\mathbf{c}} \leftarrow \mathbf{0}$ {size $|\mathcal{D}|$}, $d \leftarrow 0$, $1/v \leftarrow 0$
10:  $\mathbf{g} \leftarrow \tilde{\mathbf{K}}^{-1}\tilde{\mathbf{k}}(\mathbf{b}, a)$, $\delta \leftarrow k((\mathbf{b}, a), (\mathbf{b}, a)) - \tilde{\mathbf{k}}(\mathbf{b}, a)^\mathsf{T}\mathbf{g}$
11:  if $\delta > \nu$ then
12:    $\mathcal{D} \leftarrow \{(\mathbf{b}, a)\} \cup \mathcal{D}$, $\tilde{\mathbf{K}}^{-1} \leftarrow \frac{1}{\delta}\begin{bmatrix} \delta\tilde{\mathbf{K}}^{-1} + \mathbf{g}\mathbf{g}^\mathsf{T} & -\mathbf{g} \\ -\mathbf{g}^\mathsf{T} & 1 \end{bmatrix}$
13:    $\mathbf{g} \leftarrow [0, \ldots, 0, 1]^\mathsf{T}$ {size $|\mathcal{D}|$}, $\tilde{\boldsymbol{\mu}} \leftarrow \begin{bmatrix} \tilde{\boldsymbol{\mu}} \\ 0 \end{bmatrix}$, $\tilde{\mathbf{C}} \leftarrow \begin{bmatrix} \tilde{\mathbf{C}} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix}$, $\tilde{\mathbf{c}} \leftarrow \begin{bmatrix} \tilde{\mathbf{c}} \\ 0 \end{bmatrix}$
14:  end if
15:  for each step in the episode do
16:    Take action $a$, observe reward $r'$, update belief state $\mathbf{b}'$, choose new action
       $a' = \arg\max_{a'} \tilde{\mathbf{k}}(\mathbf{b}', a')^\mathsf{T}\tilde{\boldsymbol{\mu}}$ or randomly with probability $\epsilon$
17:    if non-terminal step then
18:      $\mathbf{g} \leftarrow \tilde{\mathbf{K}}^{-1}\tilde{\mathbf{k}}(\mathbf{b}', a')$, $\delta \leftarrow k((\mathbf{b}', a'), (\mathbf{b}', a')) - \tilde{\mathbf{k}}(\mathbf{b}', a')^\mathsf{T}\mathbf{g}$
19:      $\Delta\tilde{\mathbf{k}} \leftarrow \tilde{\mathbf{k}}(\mathbf{b}, a) - \gamma\tilde{\mathbf{k}}(\mathbf{b}', a')$
20:    else
21:      $\mathbf{g} \leftarrow [\mathbf{0}]$ {size $|\mathcal{D}|$}, $\delta \leftarrow 0$, $\Delta\tilde{\mathbf{k}} \leftarrow \tilde{\mathbf{k}}(\mathbf{b}, a)$
22:    end if
23:    $d \leftarrow \frac{\gamma\sigma^2}{v}d + r - \Delta\tilde{\mathbf{k}}^\mathsf{T}\tilde{\boldsymbol{\mu}}$
24:    if $\delta > \nu$ then
25:      $\mathcal{D} \leftarrow \{(\mathbf{b}', a')\} \cup \mathcal{D}$, $\tilde{\mathbf{K}}^{-1} \leftarrow \frac{1}{\delta}\begin{bmatrix} \delta\tilde{\mathbf{K}}^{-1} + \mathbf{g}\mathbf{g}^\mathsf{T} & -\mathbf{g} \\ -\mathbf{g}^\mathsf{T} & 1 \end{bmatrix}$
26:      $\mathbf{g} \leftarrow [0, \ldots, 0, 1]^\mathsf{T}$, $\tilde{\mathbf{h}} \leftarrow [\mathbf{g}^\mathbf{T}, -\gamma]^\mathsf{T}$
27:      $\Delta k_{tt} \leftarrow \mathbf{g}^\mathsf{T}(\tilde{\mathbf{k}}(\mathbf{b}, a) - 2\gamma\tilde{\mathbf{k}}(\mathbf{b}', a')) + \gamma^2 k((\mathbf{b}', a'), (\mathbf{b}', a'))$
28:      $\tilde{\mathbf{c}} \leftarrow \frac{\gamma\sigma^2}{v}\begin{bmatrix} \tilde{\mathbf{c}} \\ 0 \end{bmatrix} + \tilde{\mathbf{h}} - \begin{bmatrix} \tilde{\mathbf{C}}\Delta\tilde{\mathbf{k}} \\ 0 \end{bmatrix}$
29:      $v \leftarrow (1 + \gamma^2)\sigma^2 + \Delta k_{tt} - \Delta\tilde{\mathbf{k}}^\mathsf{T}\tilde{\mathbf{C}}\Delta\tilde{\mathbf{k}} + \frac{2\gamma\sigma^2}{v}\tilde{\mathbf{c}}\Delta\tilde{\mathbf{k}} - \frac{\gamma^2\sigma^4}{v}$
30:      $\tilde{\boldsymbol{\mu}} \leftarrow \begin{bmatrix} \tilde{\boldsymbol{\mu}} \\ 0 \end{bmatrix}$, $\tilde{\mathbf{C}} \leftarrow \begin{bmatrix} \tilde{\mathbf{C}} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix}$
31:    else
32:      $\tilde{\mathbf{h}} \leftarrow \mathbf{g} - \gamma\mathbf{g}$, $\tilde{\mathbf{c}} \leftarrow \frac{\gamma\sigma^2}{v}\tilde{\mathbf{c}} + \tilde{\mathbf{h}} - \tilde{\mathbf{C}}\Delta\tilde{\mathbf{k}}$
33:      if non-terminal step then
34:        $v \leftarrow (1 + \gamma^2)\sigma^2 + \Delta\tilde{\mathbf{k}}^\mathsf{T}(\tilde{\mathbf{c}} + \frac{\gamma\sigma^2}{v}\tilde{\mathbf{c}}) - \frac{\gamma^2\sigma^4}{v}$
35:      else
36:        $v \leftarrow \sigma^2 + \Delta\tilde{\mathbf{k}}^\mathsf{T}(\tilde{\mathbf{c}} + \frac{\gamma\sigma^2}{v}\tilde{\mathbf{c}}) - \frac{\gamma^2\sigma^4}{v}$
37:      end if
38:    end if
39:    $\tilde{\boldsymbol{\mu}} \leftarrow \tilde{\boldsymbol{\mu}} + \frac{\tilde{\mathbf{c}}}{v}d$, $\tilde{\mathbf{C}} \leftarrow \tilde{\mathbf{C}} + \frac{1}{v}\tilde{\mathbf{c}}\tilde{\mathbf{c}}^\mathsf{T}$, $\mathbf{b} \leftarrow \mathbf{b}'$, $a \leftarrow a'$
40:  end for
41: end for
```

References

[1] M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, K. Yu, and S. Young, "Training and evaluation of the HIS-POMDP dialogue system in noise," in *Proceedings of SIGDIAL*, 2008. i

[2] M. Gašić, F. Lefèvre, F. Jurčíček, S. Keizer, F. Mairesse, B. Thomson, K. Yu, and S. Young, "Back-off Action Selection in Summary Space-Based POMDP Dialogue Systems," in *Proceedings of ASRU*, 2009. i

[3] M. Gašić and S. Young, "Effective Handling of Dialogue State in the Hidden Information State POMDP Dialogue Manager," *ACM Transactions on Speech and Language Processing*, To appear. i

[4] ——, "Effective Handling of Dialogue State in the Hidden Information State POMDP Dialogue Manager," Cambridge Univ. Engineering Dept, Tech. Rep. CUED/F-INFENG/TR.650, 2010. i

[5] F. Weng, L. Cavedon, B. Raghunathan, D. Mirkovic, H. Cheng, H. Schmidt, H. Bratt, R. Mishra, S. Peters, L. Zhao, S. Upson, E. Shriberg, and C. Bergmann, "Developing a conversational dialogue system for cognitively overloaded users," in *Proceedings of the International Congress on Intelligent Transportation Systems*, 2004. 2

[6] N. Patel, S. Agarwal, N. Rajput, A. Nanavati, P. Dave, and T. Parikh, "Experiences designing a voice interface for rural India," in *Proceedings of SLT*, 2008. 2

[7] R. Tucker and M. Gakuru, "Experience with developing and deploying an agricultural information system using spoken language technology in Kenya," in *Proceedings of SLT*, 2008. 2

[8] S. Seneff, “Response planning and generation in the MERCURY flight reservation system,” *Computer Speech and Language*, vol. 16, pp. 283–312, 2002. 3, 12

[9] A. Raux, D. Bohus, B. Langner, A. Black, and M. Eskenazi, “Doing research on a deployed spoken dialogue system: one year of Let’s Go! experience,” in *Proceedings of ICSLP*, 2006. 3, 12

[10] J. Williams, “Applying POMDPs to dialog systems in the troubleshooting domain,” in *Proceedings of HLT*, 2007. 3, 12, 38, 60

[11] T. Paek and R. Pieraccini, “Automating spoken dialogue management design using machine learning: An industry perspective,” *Speech Communication*, vol. 50, pp. 716–729, 2008. 3, 13, 14

[12] R. P. Lippmann, “Speech recognition by machines and humans,” *Speech Communication*, vol. 22, no. 1, pp. 1–15, 1997. 3

[13] M. A. Walker, “An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email,” *Journal of Artificial Intelligence Research*, vol. 12, pp. 387–416, June 2000. 3

[14] A. M. Turing, *Computing machinery and intelligence.* MIND, 1950. 3

[15] O. Lemon and O. Pietquin, “Machine Learning for Spoken Dialogue Systems,” in *Proceedings of Interspeech*, 2007. 3

[16] D. Bohus and E. Horvitz, “Models for Multiparty Engagement in Open-World Dialog,” in *Proceedings of SIGDIAL*, 2009. 3

[17] A. Rudnicky and W. Xu, “An agenda-based dialog management architecture for spoken language systems,” in *Proceedings of ASRU*, 1999. 3

[18] M. McTear, *Spoken Dialogue Technology.* London, UK: Springer, 2004. 4, 11, 12, 13

[19] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000. 4

[20] S. Young, "Talking to Machines (Statistically Speaking)," in *Proceedings of ICSLP*, 2002. 4, 32, 41

[21] N. Roy, J. Pineau, and S. Thrun, "Spoken dialogue management using probabilistic reasoning," in *Proceedings of ACL*, 2000. 4, 32, 33, 36

[22] B. Zhang, Q. Cai, J. Mao, and B. Guo, "Planning and Acting under Uncertainty: A New Model for Spoken Dialogue System," in *Proceedings of Conference on Uncertainty in Artificial Intelligence*, 2001. 4

[23] J. Williams and S. Young, "Partially Observable Markov Decision Processes for Spoken Dialog Systems," *Computer Speech and Language*, vol. 21, no. 2, pp. 393–422, 2007. 4

[24] R. Schluter and W. Macherey, "Comparison of discriminative training criteria," in *Proceedings of ICASSP*, 1998. 5

[25] W. Chou, C. Lee, and B. Juang, "Minimum error rate training based on n-best string models," in *Proceedings of ICASSP*, 1993. 5

[26] F. Wessel, K. Schlutr, K. Macherey, and H. Ney, "Confidence measures for large vocabulary continuous speech recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 9, no. 3, pp. 288–298, 2001. 5

[27] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. 5

[28] M. Gales and S. Young, "The application of hidden Markov models in speech recognition," *Found. Trends Signal Process.*, vol. 1, pp. 195–304, 2007. 5, 142

[29] A. Black, H. Zen, and K. Tokuda, "Statistical parametric speech synthesis," in *Proceedings of ICASSP*, 2007. 6

[30] F. Mairesse, M. Gašić, F. Jurčíček, S. Keizer, B. Thomson, K. Yu, and S. Young, "Spoken language understanding from unaligned data using discriminative classification models," in *Proceedings of ICASSP*, 2009. 6, 42

[31] F. Mairesse, M. Gašić, F. Jurčíček, S. Keizer, B. Thomson, K. Yu, and S. Young, "Phrase-based statistical language generation using graphical models and active learning," in *Proceedings of ACL*, 2010. 6

[32] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, ser. Adaptive Computation and Machine Learning. Cambridge, Massachusetts: MIT Press, 1998. 6, 15, 17, 18, 19, 20, 60, 114, 166

[33] E. Levin, R. Pieraccini, and W. Eckert, "Using Markov Decision Processes for Learning Dialogue Strategies," in *Proceedings of ICASSP*, 1998. 6, 15, 23

[34] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and Acting in Partially Observable Stochastic Domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998. 6, 27, 28, 31

[35] S. Young, J. Schatzmann, K. Weilhammer, and H. Ye, "The Hidden Information State Approach to Dialog Management," in *Proceedings of ICASSP*, 2007. 7, 41

[36] J. Williams and S. Young, "Scaling up POMDPs for Dialogue Management: the Summary POMDP Method," in *Proceedings of ASRU*, 2005. 7

[37] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Cambridge, Massachusetts: MIT Press, 2005. 9, 115, 116, 121, 126, 130

[38] O. Lemon, K. Georgila, and J. Henderson, "Evaluating Effectiveness and Portability of Reinforcement Learned Dialogue Strategies with real users: the TALK TownInfo Eval," in *Proceedings of SLT*, 2006. 12, 38

[39] A. Black and M. Eskenazi, "The spoken dialogue challenge," in *Proceedings of SIGDIAL*, 2009. 12

[40] A. Black, S. Burger, B. Langner, G. Parent, and M. Eskenazi, "Spoken Dialog Challenge 2010," in *Proceedings of SLT*, 2010. 12

[41] B. Mazor and B. L. Zeigler, "The design of speech-interactive dialogs for transaction-automation systems," *Speech Communication*, vol. 17, no. 3-4, pp. 313–320, 1995. 13

[42] M. McTear, "Modelling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit," in *Proceedings of ICSLP*, 1998. 13

[43] R. Pieraccini and J. Huerta, "Where do we go from here? Research and commercial spoken dialog systems," in *Proceedings of SIGDIAL*, 2005. 13, 14

[44] M. Cohen, J. Giangola, and J. Balough, *Voice User Interface Design.* Addison Wesley, 2004. 13

[45] S. McGlashan, D. Burnett, J. Carter, P. Danielsen, J. Ferrans, A. Hunt, B. Lucas, B. Porter, K. Rehor, and S. Tryphonas, "Voice Extensible Markup Language (VoiceXML) Version 2.0," W3C, Technical specification, 2004. 13

[46] D. Goddeau, H. Meng, J. Polifroni, S. Seneff, and S. Busayapongchaiy, "A Form-Based Dialogue Manager For Spoken Language Applications," in *Proceedings of ICSLP*, 1996. 13

[47] D. Bohus and A. Rudnicky, "RavenClaw: Dialog management using hierarchical task decomposition and an expectation agenda," in *Proceedings of Eurospeech*, 2003. 13

[48] D. Bohus and A. I. Rudnicky, "Error handling in the RavenClaw dialog management framework," in *Proceedings of HLT*, 2005. 14

[49] J. R. Glass, "Challenges For Spoken Dialogue Systems," in *Proceedings of ASRU*, 1999. 14

[50] R. Pieraccini, D. Suendermann, K. Dayanidhi, and J. Liscombe, "Are We There Yet? Research in Commercial Spoken Dialog Systems," in *Proceedings of TSD*, 2009. 14

[51] E. Levin, R. Pieraccini, and W. Eckert, "A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 1, pp. 11–23, 2000. 15, 23, 24, 37, 38

[52] R. E. Bellman, *Dynamic Programming.* Princeton University Press, Princeton, 1957. 17, 18

[53] J. Peters and S. Schaal, “Natural Actor-Critic,” *Neurocomputing*, vol. 71, pp. 1180–1190, 2008. 23, 36

[54] S. Singh, M. Kearns, D. Litman, and M. Walker, “Reinforcement Learning for Spoken Dialogue System,” in *Proceedings of NIPS*, 1999. 23, 24, 25

[55] S. Young, “Probabilistic Methods in Spoken Dialogue Systems,” *Philosophical Trans Royal Society (Series A)*, vol. 358, no. 1769, pp. 1389–1402, 2000. 23, 24

[56] D. J. Litman, M. S. Kearns, S. Singh, and M. A. Walker, “Automatic optimization of dialogue management,” in *Proceedings of ACL*, 2000. 23, 25

[57] D. Goddeau and J. Pineau, “Fast reinforcement learning of dialog strategies,” in *Proceedings of ICASSP*, 2000. 23, 24

[58] S. Singh, D. Litman, M. Kearns, and M. Walker, “Optimizing Dialogue Management with Reinforcement Learning,” *Journal of Artificial Intelligence Research*, vol. 16, pp. 105–133, 2002. 23, 25, 38, 142

[59] K. Georgila and O. Lemon, “Adaptive multimodal dialogue management based on the information state update approach,” in *W3C Workshop on Multimodal Interaction*, 2004. 23, 142

[60] S. P. Singh, M. J. Kearns, D. J. Litman, and M. A. Walker, “Empirical Evaluation of a Reinforcement Learning Spoken Dialogue System,” in *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 2000. 23

[61] H. Cuayhuitl, S. Renals, O. Lemon, and H. Shimodaira, “Learning Multi-Goal Dialogue Strategies Using Reinforcement Learning With Reduced State-Action Spaces,” in *International Journal of Game Theory*, 2006, pp. 547–565. 24

[62] J. Henderson, O. Lemon, and K. Georgila, “Hybrid Reinforcement/Supervised Learning for Dialogue Policies from COMMUNICATOR Data,” in *Proceedings of IJCAI*, 2005. 24

[63] V. Rieser, I. Kruijff-Korbayová, and O. Lemon, "A corpus collection and annotation framework for learning multimodal clarification strategies," in *Proceedings of SIGDIAL*, 2005. 24

[64] N. Fraser and G. Gilbert, "Simulating speech systems," *Computer Speech and Language*, vol. 5, no. 1, pp. 81–99, 1991. 25

[65] V. Reiser, "Bootstrapping Reinforcement Learning-based Dialogue Strategies from Wizard-of-Oz data," Ph.D. dissertation, Universität des Saarlandes, Saarbrüeken, 2008. 25

[66] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proceedings of IJCAI*, 2003, pp. 1025–1032. 31

[67] J. Williams and S. Young, "Scaling POMDPs for Spoken Dialog Management," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 7, pp. 2116–2129, 2007. 31

[68] R. Brafman, "A Heuristic Variable Grid Solution Method for POMDPs," in *AAAI*, 1997. 31

[69] B. Zhang, Q. Cai, J. Mao, E. Chang, and B. Guo, "Spoken Dialogue Management as Planning and Acting under Uncertainty," in *Proceedings of Eurospeech*, 2001. 32, 33, 36

[70] J. Williams, "Partially Observable Markov Decision Processes for Spoken Dialogue Management," Ph.D. dissertation, University of Cambridge, 2006. 32, 128

[71] T. Bui, B. van Schooten, and D. Hofs, "Practical dialogue manager development using POMDPs," in *Proceedings of SIGDIAL*, 2007. 32, 35, 36

[72] K. Kim, C. Lee, S. Jung, and G. Lee, "A frame-based probabilistic framework for spoken dialog management using dialog examples," in *Proceedings of SIGDIAL*, 2008. 32, 73

[73] B. Thomson, "Statistical methods for spoken dialogue management," Ph.D. dissertation, University of Cambridge, 2009. 32, 36, 59, 63, 114, 142, 163

[74] S. Varges, S. Quarteroni, G. Riccardi, and A. V. Ivanov, "Investigating clarification strategies in a hybrid POMDP dialog manager," in *Proceedings of SIGDIAL*, 2010. 32

[75] J. Williams, P. Poupart, and S. Young, "Factored Partially Observable Markov Decision Processes for Dialogue Management," in *Proceedings of Knowledge and Reasoning in Practical Dialogue Systems*, 2005. 33, 34, 44, 46

[76] S. Young, M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu, "The Hidden Information State model: A practical framework for POMDP-based spoken dialogue management," *Computer Speech and Language*, vol. 24, no. 2, pp. 150–174, 2010. 35, 36, 41

[77] B. Thomson and S. Young, "Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems," *Computer Speech and Language*, vol. 24, no. 4, pp. 562–588, 2010. 35, 36, 37, 38, 60, 62

[78] J. Williams, "Using particle filters to track dialogue state," in *Proceedings of ASRU*, 2007. 35

[79] B. Thomson, K. Yu, M. Gašić, F. Jurčíček, F. Mairesse, and S. Young, "Bayesian Dialogue System for the Let's Go Spoken Dialogue Challenge," in *Proceedings of SLT*, 2010. 36

[80] F. Pinaut, F. Lefèvre, and R. De Mori, "Feature-based Summary Space for Stochastic Dialogue Modelling with Hierarchical Semantic Frames," in *Proceedings of Interspeech*, 2009. 36

[81] S. Amari, "Natural gradient works efficiently in learning," *Neural computation*, vol. 10, pp. 251–276, 1998. 36

[82] L. Li, J. Williams, and S. Balakrishnan, "Reinforcement Learning for Dialog Management using Least-Squares Policy Iteration and Fast Feature Selection," in *Proceedings of Interspeech*, 2009. 37

[83] Y. Minami, R. Higashinaka, K. Dohsaka, T. Meguro, and E. Maeda, "Trigram dialogue control using POMDPs," in *Proceedings of SLT*, 2010. 37

[84] K. Scheffler and S. Young, “Corpus-based dialogue simulation for automatic strategy learning and evaluation,” in *Proceedings of NAACL*, 2001. 37

[85] K. Georgila, J. Henderson, and O. Lemon, “Learning User Simulations for Information State Update Dialog Systems,” in *Proceedings of Eurospeech*, 2005. 37

[86] J. Schatzmann, K. Weilhammer, M. Stuttle, and S. Young, “A Survey of Statistical User Simulation Techniques for Reinforcement-Learning of Dialogue Management Strategies,” *KER*, vol. 21, no. 2, pp. 97–126, 2006. 37

[87] V. Rieser and O. Lemon, “Cluster-based User Simulations for Learning Dialogue Strategies,” in *Proceedings of Interspeech*, 2006. 37

[88] S. Quarteroni, M. González, G. Riccardi, and S. Varges, “Combining user intention and error modeling for statistical dialog simulators,” in *Proceedings of Interspeech*, 2010. 37

[89] K. Scheffler and S. Young, “Probabilistic simulation of human-machine dialogues,” in *ICASSP*, 2000. 37

[90] J. Schatzmann, “Statistical User and Error Modelling for Spoken Dialogue Systems,” Ph.D. dissertation, University of Cambridge, 2008. 37, 38, 59, 60, 61, 62, 63, 66

[91] K. Georgila, J. Henderson, and O. Lemon, “User Simulation for Spoken Dialogue Systems: Learning and Evaluation,” in *Proceedings of Interspeech*, 2006. 37

[92] O. Pietquin and T. Dutoit, “A probabilistic framework for dialog simulation and optimal strategy learning,” *IEEE Transactions on Speech and Audio Processing, Special Issue on Data Mining of Speech, Audio and Dialog*, vol. 14, no. 2, pp. 589–599, 2006. 37

[93] J. Schatzmann and S. Young, “The Hidden Agenda User Simulation Model,” *IEEE Transactions on Speech and Audio Processing*, vol. 17, no. 4, pp. 733–747, 2009. 37

[94] S. Keizer, M. Gašić, F. Jurčíček, F. Mairesse, B. Thomson, K. Yu, and S. Young, "Parameter estimation for agenda-based user simulation," in *Proceedings of SIGDIAL*, 2010. 37, 61, 142

[95] L. Hirschman and H. S. Thompson, *Overview of evaluation in speech and natural language processing.* New York, NY, USA: Cambridge University Press, 1997, pp. 409–414. 38

[96] S. Gandhe and D. Traum, "An evaluation understudy for dialogue coherence models," in *Proceedings of SIGDIAL*, 2008. 38

[97] T. Watambe, M. Araki, and S. Doshita, "Evaluating Dialogue Strategies under Communication Errors Using Computer-to-Computer Simulation ," *IEICE Trans. Inf. and Syst.*, vol. E81-D, no. 9, pp. 1025–1033, 1998. 38

[98] H. Ai and F. Weng, "User simulation as testing for spoken dialog systems," in *Proceedings of SIGDIAL*, 2008. 38

[99] M. A. Walker, D. J. Litman, C. A. Kamm, and A. Abella, "PARADISE: a framework for evaluating spoken dialogue agents," in *Proceedings of EACL*, 1997. 38

[100] S. Young, J. Williams, J. Schatzmann, M. Stuttle, and K. Weilhammer, "The Hidden Information State Approach to Dialogue Management," Cambridge Univ. Engineering Dept, Tech. Rep. CUED/F-INFENG/TR.544, 2005. 41

[101] S. Young, "ATK: An Application Toolkit for HTK," http://mi.eng.cam.ac.uk/research/dialogue/atk_home, 2005. 42

[102] B. Thomson, K. Yu, M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, and S. Young, "Evaluating semantic-level confidence scores with multiple hypotheses," in *Proceedings of Interspeech*, 2008. 42, 73

[103] K. Yu, T. Toda, M. Gašić, S. Keizer, F. Mairesse, B. Thomson, and S. Young, "Probablistic modelling of F0 in unvoiced regions in HMM based speech synthesis," in *Proceedings of ICASSP*, 2009. 42

[104] J. Allen and M. Core, "Draft of DAMSL: Dialog Act Markup in Several Layers," Dagstuhl Workshop, 1997. [Online]. Available: http://www.cs.rochester.edu/research/cisd/resources/damsl/ 44

[105] D. Traum, “Computational Models of Grounding in Collaborative Systems,” in *Working Papers of the AAAI Fall Symposium on Psychological Models of Communication in Collaborative Systems*, 1999, pp. 124–131. 50

[106] S. Keizer, M. Gašić, F. Mairesse, B. Thomson, K. Yu, and S. Young, “Modelling User Behaviour in the HIS-POMDP Dialogue Manager,” in *Proceedings of SLT*, 2008. 53

[107] J. Schatzmann, B. Thomson, and S. Young, “Error Simulation for Training Statistical Dialogue Systems,” in *Proceedings of ASRU*, 2007. 61

[108] W. Ward, “The Phoenix system: Understanding spontaneous speech,” in *Proceedings of ICASSP*, 1991. 66

[109] A. Black and K. Lenzo, “Flite: a small, fast speech synthesis engine,” 2005, http://www.speech.cs.cmu.edu/flite/doc. 66

[110] J. Williams, “Incremental Partition Recombination for Efficient Tracking of Multiple Dialogue States,” in *Proceedings of ICASSP*, 2010. 73, 87, 89, 94, 97, 162

[111] R. Reiter, “A logic for default reasoning,” *Artificial Intelligence*, vol. 13, no. 1–2, pp. 81–132, 1980. 84

[112] J. Groenendijk and M. Stokhof, “Changing the context: Dynamics and discourse,” in *Proceedings of IATL*, 1996. 84, 85

[113] S. Levinson, *Pragmatics.* Cambridge: Cambridge University Press, 1983. 84, 85

[114] F. Veltman, “Defaults in update semantics,” *Journal of Philosophical Logic*, 1996. 85

[115] A. Lascarides and N. Asher, “Segmented Discourse Representation Theory: Dynamic Semantics with Discourse Structure,” *Computing Meaning*, vol. 3, 2001. 85, 86

[116] H. Kamp and U. Reyle, *From Discourse to the Lexicon: Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and*

Discourse Representation Theory. Kluwer Academic Publishers, 1993. 85

[117] J. Henderson and O. Lemon, "Mixture model POMDPs for efficient handling of uncertainty in dialogue management," in *Proceedings of ACL*, 2008. 87

[118] M. Deisenroth, C. Rasmussen, and J. Peters, "Gaussian Process Dynamic Programming," *Neurocomputing*, vol. 72, no. 7-9, pp. 1508–1524, 2009. 114, 128, 167

[119] Y. Engel, S. Mannor, and R. Meir, "Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning," in *Proceedings of ICML*, 2003. 114

[120] ——, "Reinforcement learning with Gaussian processes," in *Proceedings of ICML*, 2005. 114, 117, 120, 125, 127

[121] C. E. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 16. MIT Press, 2004, pp. 751–759. 114

[122] M. Stein, *Interpolation of spatial data: some theory for kriging*, ser. Springer series in statistics. New York: Springer, 1999. 121

[123] J. Quinonero-Candela and C. Rasmussen, "A Unifying View of Sparse Approximate Gaussian Process Regression," *Journal of Machine Learning Research*, vol. 6, pp. 1939–1959, 2005. 122, 126, 133, 167

[124] E. Snelson and Z. Ghahramani, "Sparse Gaussian Processes using Pseudo-inputs," in *Proceedings of NIPS*, 2006. 122, 167

[125] Y. Engel, "Algorithms and Representations for Reinforcement Learning," PhD thesis, Hebrew University, 2005. 123, 124, 167, 198, 199, 200

[126] B. Scholkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* Cambridge, MA: MIT Press, 2001. 125

[127] M. Lázaro-Gredilla, J. Quiñonero Candela, C. Rasmussen, and A. Figueiras-Vidal, “Sparse Spectrum Gaussian Process Regression,” *Journal of Machine Learning Research*, vol. 11, pp. 1865–1881, 2010. 126

[128] MacKay, D.J.C., “Information-based objective functions for active data selection,” *Neural Computation*, vol. 4, no. 4, pp. 590–604, 1992. 127

[129] D. Cohn, L. Atlas, and R. Ladner, “Improving Generalization with Active Learning,” *Machine Learning*, vol. 15, pp. 201–221, 1994. 127

[130] S. Dasgupta and D. Hsu, “Hierarchical sampling for active learning,” in *Proceedings of ICML*, 2008. 127

[131] F. Doshi, J. Pineau, and N. Roy, “Reinforcement learning with limited reinforcement: using Bayes risk for active learning in POMDPs,” in *Proceedings of ICML*, 2008. 128

[132] R. M. Neal, *Bayesian Learning for Neural Networks.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1996. 129

[133] C. Rasmussen and C. Williams, “Software for Gaussian Processes Regression and Classification,” 2007, http://www.gaussianprocess.org/gpml/code/matlab/doc/. 130

[134] A. Yaglom, *Correlation Theory of Stationary and Related Random Functions Volume I: Basic Results.* New York: Springer-Verlag, 1987. 131

[135] A. Cassandra, “POMDP solver,” 2005, http://www.pomdp.org. 131

[136] M. Collins and N. Duffy, “New ranking algorithms for parsing and tagging: kernels over discrete structures, and the voted perceptron,” in *Proceedings of ACL*, 2002. 134, 166

[137] C. Cortes, P. Haffner, M. Mohri, K. Bennett, and N. Cesa-bianchi, “Rational kernels: Theory and algorithms,” *Journal of Machine Learning Research*, vol. 5, pp. 1035–1062, 2004. 135, 166

[138] J. Reisinger, P. Stone, and R. Miikkulainen, "Online kernel selection for Bayesian reinforcement learning," in *Proceedings of ICML*, 2008. 137

[139] D. J. Litman and S. Pan, "Empirically evaluating an adaptable spoken dialogue system," in *Proceedings of the seventh international conference on User modelling*, 1999. 142

[140] D. J. Litman and S. Pany, "Designing and evaluating an adaptive spoken dialogue system," *User Modelling and User-Adapted Interaction*, vol. 12, pp. 111–137, 2002. 142

[141] S. Janarthanam and O. Lemon, "Adaptive Referring Expression Generation in Spoken Dialogue Systems: Evaluation with Real Users," in *Proceedings of SIGDIAL*, 2010. 142, 143, 145

[142] M. Gales, "Cluster adaptive training for speech recognition," in *Proceedings of ICSLP*, 1998, pp. 1783–1786. 144

[143] P. Dallaire, C. Besse, S. Ross, and B. Chaib-draa, "Bayesian reinforcement learning in continuous pomdps with gaussian processes," in *Proceedings of IROS*, 2009. 167

[144] C. Lee, S. Narayanan, and R. Pieraccini, "Classifying emotions in human-machine spoken dialogs," in *Proceedings of ICME*, 2002. 168

[145] F. Jurčíček, S. Keizer, M. Gašić, F. Mairesse, B. Thomson, K. Yu, and S. Young, "Real user evaluation of spoken dialogue systems using Amazon Mechanical Turk," in *Proceedings of Interspeech*, submitted. 168

[146] L. Scharf, *Statistical Signal Processing.* Reading, Massachusetts: Addison-Wesley, 1991. 200