
**Partially Observable
Markov Decision Processes
for
Spoken Dialogue Management**

Jason D. Williams

Churchill College
and
Cambridge University Engineering Department



April 2006

This dissertation is submitted for the degree of Doctor of Philosophy
to the University of Cambridge

Declaration

This dissertation is the result of my own work carried out at the Cambridge University Engineering Department and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text. Some material has been previously presented at international conferences [112] [122] [123] [125] [126].

The length of this dissertation, including appendices, bibliography, footnotes, tables and equations is approximately 47,000 words. This dissertation contains 77 figures (labelled in the text as “Figures” and “Algorithms”).

Abstract

Partially Observable Markov Decision Processes for Spoken Dialogue Management

Jason D. Williams

The design of robust spoken dialog systems is a significant research challenge. Speech recognition errors are common and hence the state of the conversation can never be known with certainty, and users can react in a variety of ways making deterministic forward planning impossible. This thesis argues that a partially observable Markov decision process (POMDP) provides a principled formalism for modelling human-machine conversation. Further, this thesis introduces the SDS-POMDP framework which enables statistical models of users' behavior and the speech recognition process to be combined with handcrafted heuristics into a single framework that supports global optimization. A combination of theoretical and empirical studies confirm that the SDS-POMDP framework unifies and extends existing techniques, such as local use of confidence score, maintaining parallel dialog hypotheses, and automated planning.

Despite its potential, the SDS-POMDP model faces important scalability challenges, and this thesis next presents two methods for scaling up the SDS-POMDP model to realistically sized spoken dialog systems. First, summary point-based value iteration (SPBVI) enables a single *slot* (a dialog variable such as a date, time, or location) to take on an arbitrary number of values by restricting the planner to consider only the likelihood of the best hypothesis. Second, composite SPBVI (CSPBVI) enables dialog managers consisting of *many slots* to be created by planning *locally* within each slot, and combining these local plans into a global plan using a simple heuristic. Results from dialog simulation show that these techniques enable the SDS-POMDP model to handle real-world dialog problems while continuing to out-perform established techniques and hand-crafted dialog managers. Finally, application to a real spoken dialog system is demonstrated.

Keywords: Spoken dialog systems, dialog management, partially observable Markov decision processes, decision theory.

Acknowledgements

First I would like to thank my supervisor Steve Young, who has invested continuous and substantial thought, time, and trust; and from whom I have learned a great deal about how to think, listen, and communicate.

Thanks also to Pascal Poupart for encouragement, mentoring, and many insightful discussions. Thanks to my lab group – Jost Schatzmann, Matt Stuttle, Hui “KK” Ye, and Karl Weilhammer – for motivating discussions and for fun distractions. Thanks also to Patrick Gosling and Anna Langley for skillfully keeping the Fallside lab running.

Thanks to the “Dialogs on Dialogs” group at Carnegie Mellon University for providing a critical forum of peers. I have also benefitted from a great many conversations and email exchanges with others in the field, too numerous to list here.

I am deeply appreciative of the financial support I have received from the Gates Cambridge Trust as a Gates Scholar, and from the Government of the United Kingdom as an Overseas Research Students Award recipient. I am also grateful of the additional support I have received from the European Union Framework 6 TALK project, from the Department of Engineering, and from Churchill College.

Finally, I owe a great debt of thanks to my wife, Rosemary, for unfailing support; for encouraging me to pursue a PhD in the first place; and for many, many wonderful cups of fruit tea, lovingly served.

Contents

List of Figures	vii
List of Tables	x
List of Algorithms	xi
Notation and Acronyms	xii
1 Introduction	1
2 POMDP Background	5
2.1 Introduction to POMDPs	5
2.2 Finding POMDP policies	9
2.3 Value iteration	14
2.4 Point-based value iteration	20
3 Dialog management as a POMDP	25
3.1 Components of a spoken dialog system	25
3.2 The SDS-POMDP model	27
3.3 Example SDS-POMDP application: TRAVEL	30
3.4 Optimization of the TRAVEL application	35
4 Comparisons with existing techniques	41
4.1 Automated planning	43
4.2 Local confidence scores	48
4.3 Parallel state hypotheses	54
4.4 Handcrafted dialog managers	58
4.5 Other POMDP-based dialog managers	60
4.6 Chapter summary	63

5	Scaling up: Summary point-based value iteration (SPBVI)	64
5.1	Slot-filling dialogs	64
5.2	SPBVI method description	66
5.3	Example SPBVI application: MAXITRAVEL	73
5.4	Comparisons with baselines	79
6	Scaling up: Composite SPBVI (CSPBVI)	86
6.1	CSPBVI method description	86
6.2	Example CSPBVI application: MAXITRAVEL-W	91
6.3	Comparisons with baselines	96
6.4	Application to a practical spoken dialog system	105
7	Conclusions and future work	112
7.1	Thesis summary	112
7.2	Future work	113
	Bibliography	117

List of Figures

1.1	High-level architecture of a spoken dialog system.	1
2.1	Example of belief space.	7
2.2	Illustration of the belief monitoring process.	10
2.3	Example 3-step conditional plans.	11
2.4	Example calculation of a value function.	12
2.5	Example value function.	13
2.6	Illustration of value iteration: first step.	16
2.7	Illustration of value iteration: second step (before pruning).	17
2.8	Illustration of value iteration: second step (after pruning).	18
2.9	Illustration of value iteration: Terminal step.	18
2.10	Example conversation using an optimal policy.	19
2.11	Example PBVI policy.	23
3.1	Typical architecture of a spoken dialog system.	26
3.2	SDS-POMDP model shown as an influence diagram (detailed).	30
3.3	Probability densities for various levels of confidence score informativeness.	34
3.4	Number of belief points vs. average return (TRAVEL).	37
3.5	Number of iterations of PBVI vs. average return (TRAVEL).	38
3.6	Concept error rate vs. average return and average dialog length (TRAVEL).	39
3.7	Example conversation between user and POMDP dialog controller (TRAVEL).	40
4.1	SDS-POMDP model shown as an influence diagram (overview).	42
4.2	Dialog model based on supervised learning (influence diagram).	44
4.3	Dialog model based on a Markov Decision Process (influence diagram).	44
4.4	Benefit of maintaining multiple dialog state hypotheses (example conversation).	46
4.5	Benefit of an integrated user model (example conversation).	47
4.6	Concept error rate vs. average return for POMDP and MDP baseline.	49

4.7	Dialog model with confidence score (influence diagram).	50
4.8	Illustration of a high-confidence recognition (sample dialog).	51
4.9	Illustration of a low-confidence recognition (sample dialog).	52
4.10	Concept error rate vs. average return for the POMDP and MDP-2 baseline.	54
4.11	Confidence score informativeness vs. average return for the POMDP and MDP-2 baseline.	55
4.12	Dialog model with greedy decision theoretic policy (influence diagram).	56
4.13	Dialog model with multiple state hypotheses (influence diagram).	57
4.14	Greedy vs. POMDP policies for the VOICEMAIL application.	58
4.15	Concept error rate vs. average return for POMDP and greedy decision theoretic policies.	59
4.16	HC1 handcrafted dialog manager baseline.	61
4.17	Concept error rate vs. average return for POMDP and 3 handcrafted baselines.	62
5.1	Number of belief points vs. average return.	77
5.2	Number of belief points vs. average return for various levels of confidence score informativeness.	77
5.3	Number of observation samples vs. average return for various concept error rates.	78
5.4	Number of observation samples vs. average return for various levels of confidence score informativeness.	79
5.5	Sample conversation with SPBVI-based dialog manager (1 of 2).	80
5.6	Sample conversation with SPBVI-based dialog manager (2 of 2).	81
5.7	Concept error rate vs. average return for SPBVI and MDP baselines.	82
5.8	Confidence score informativeness vs. average return for SPBVI and MDP-2 baseline.	83
5.9	The HC4 handcrafted dialog manager baseline.	83
5.10	The HC5 handcrafted dialog manager baseline.	83
5.11	Concept error rate vs. average return for SPBVI and handcrafted baselines.	84
5.12	POMDP size vs. average return for SPBVI and PBI baseline.	85
6.1	Sample conversation with CSPBVI-based dialog manager (first half).	97
6.2	Sample conversation with CSPBVI-based dialog manager (second half).	98
6.3	Concept error rate vs. average return for CSPBVI and SPBVI baseline.	99
6.4	Concept error rate vs. average return for CSPBVI and MDP baselines.	101
6.5	Number of slots vs. average return for CSPBVI and two MDP baselines.	102
6.6	Number of slots vs. average return for CSPBVI and MDP-2 baseline (1 of 3).	102
6.7	Number of slots vs. average return for CSPBVI and MDP-2 baseline (2 of 3).	103
6.8	Number of slots vs. average return for CSPBVI and MDP-2 baseline (3 of 3).	103
6.9	Number of slots vs. average return for CSPBVI and Handcrafted controllers.	104
6.10	Concept error rate vs. average return for training and testing user models.	105
6.11	Screen shot of the TOURISTTICKETS application (1 of 2).	107
6.12	Screen shot of the TOURISTTICKETS application (2 of 2).	108

6.13 Sample conversation with TOURISTTICKETS spoken dialog system (first half).	109
6.14 Sample conversation with TOURISTTICKETS spoken dialog system (second half).	110

List of Tables

2.1	Transition function for the VOICEMAIL application	8
2.2	Observation function for the VOICEMAIL application	8
2.3	Reward function for the VOICEMAIL application	9
2.4	Observation function for the VOICEMAIL-2 POMDP	23
3.1	Summary of SDS-POMDP components	29
3.2	Machine actions in the TRAVEL application.	31
3.3	User actions in the TRAVEL application.	31
3.4	User model parameters for the TRAVEL application.	32
3.5	Reward function for the TRAVEL application.	36
4.1	Node subscripts used in influence diagrams.	42
4.2	Listing of MDP states for MDP baseline.	48
4.3	Listing of MDP states for MDP-2 baseline.	53
5.1	Example slot-filling dialog domains.	65
5.2	User actions in the MAXITRAVEL application.	74
5.3	Machine actions in the MAXITRAVEL application.	74
5.4	User model parameters for the MAXITRAVEL application.	75
5.5	Reward function for the MAXITRAVEL application.	76
6.1	Example user actions and interpretations used in MAXITRAVEL-W	93
6.2	User model parameters for the MAXITRAVEL-W application.	94
6.3	Slot names and example values for the TOURISTTICKETS application.	106

List of Algorithms

1	Value iteration.	15
2	Belief point selection for PBVI.	21
3	Point-based value iteration (PBVI)	22
4	Function <i>bToSummary</i> for SPBVI	67
5	Function <i>aToMaster</i> for SPBVI	67
6	Function <i>sampleCorner</i> for SPBVI	68
7	Belief point selection for SPBVI.	70
8	Function <i>samplePoint</i> for SPBVI	71
9	Summary point-based value iteration (SPBVI)	72
10	SPBVI action selection procedure, used at runtime.	72
11	Function <i>aToMasterComposite</i> for CSPBVI	88
12	Function <i>bToSummaryComposite</i> for CSPBVI	88
13	Belief point selection for CSPBVI.	89
14	Function <i>samplePointComposite</i> for CSPBVI	90
15	Composite summary point-based value iteration (CSPBVI)	90
16	CSPBVI action selection procedure, used at runtime.	91
17	Function <i>chooseActionHeuristic</i> for MAXITRAVEL-W	95

Notation and acronyms

The following basic notation is used in this thesis:

$P(a)$	the discrete probability of event a , the probability mass function
$p(x)$	the probability density function for a continuous variable x
a'	the value of the variable a at the next time-step
\mathcal{A}	a finite set of elements
\mathbb{B}	alternate notation for finite set of elements
$ \mathcal{A} $	the cardinality of set \mathcal{A} (i.e., the number of elements in \mathcal{A})
$\{a_1, a_2, \dots, a_N\}$	a finite set defined by the elements composing the set
$\{a_n : 1 \leq n \leq N\}$	alternate notation for a finite set defined by the elements composing the set
$\{a_n\}$	short-hand for $\{a_n : 1 \leq n \leq N\}$, where $1 \leq n \leq N$ can be inferred
$a \in \mathcal{A}$	indicates that a is a member of the set \mathcal{A}
\sum_a	short-hand for $\sum_{a \in \mathcal{A}}$
\mathcal{A}^N	a set composed of all possible N-vectors $\{(b_1, b_2, \dots, b_N) : b_n \in \mathcal{A}\}$
$\sup \mathcal{X}$	supremum: least value greater than or equal to every element in \mathcal{X}
$y \leftarrow x$	assignment of y to value of x
$\text{randInt}(a)$	Uniformly-distributed random integer in the range $[1, a]$
$\text{randIntOmit}(a, b)$	Same as $\text{randInt}(a)$ but the distribution used excludes b
$\text{sampleDist}_a(P)$	Random index a using the probability mass function $P(a)$

Commonly-used acronyms include

ASR	Automatic speech recognition
FSA	Finite state automaton
MDP	Markov decision Process
NLU	Natural language understanding
PBVI	Point-based value iteration
POMDP	Partially observable Markov decision process
SDS	Spoken dialog system
TTS	Text-to-speech

Acronyms defined in this thesis are

CSPBVI	Composite summary point-based value iteration
SDS-POMDP	Spoken dialog system partially observable Markov decision process model
SPBVI	Summary point-based value iteration

Introduction

For computers, participating in a spoken conversation with a person is difficult. To start, automatic speech recognition (ASR) technology is error-prone, so a computer must regard everything it hears with suspicion. Worse, conflicting evidence doesn't always indicate a speech recognition error, because sometimes people change their objective in the middle of a conversation. Finally, conversation is a temporal process in which users behave in non-deterministic ways, and in which actions have both immediate and long-term effects. Machine control in this environment is a challenging engineering problem, and is the focus of this thesis.

Broadly, a spoken dialog system has three modules, one each for input, output, and control, shown in Figure 1.1. The input module converts an acoustic speech signal into a string of meaningful concepts or intentions, such as *request(flight, from(london))*; the control module maintains an internal state and decides what action to take, such as *ask(departure-date)*; and the output module renders communicative actions from the machine as audio to the user. The control module is the focus of this thesis, but it is useful to briefly sketch all three modules. Interested readers are referred to introductory texts such as [50, 108, 7, 72], survey papers such as [130, 134, 33], or example implementations such as [3, 93, 81] for additional details and references.

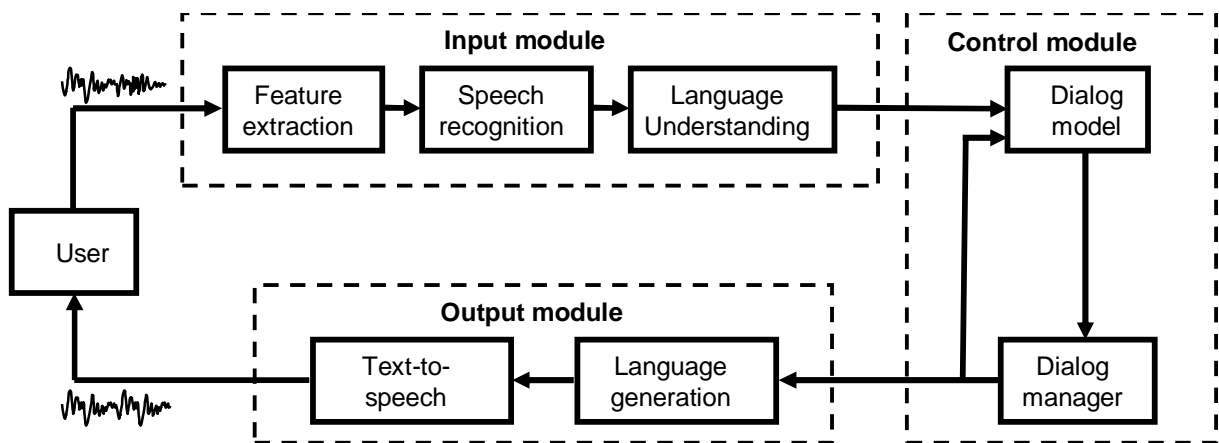


Figure 1.1 High-level architecture of a spoken dialog system.

Conceptually, the input module can be thought of as a pipeline. The user's speech arrives in the form of an *audio signal*, which is quantized, sampled, and passed to a *feature extractor*. The feature extractor considers sequential windows of speech to produce a series of *feature vectors* describing the speech signal. These feature vectors are then passed to the *speech recognizer*, which employs *acoustic models* (typically hidden Markov models [89]) to hypothesize a string of *phonemes*, the basic aural building blocks of a spoken language. A *lexicon*, which indicates how words are pronounced, is used to map from the hypothesized phoneme string to a hypothesized word string. A *language understanding* process or parser then extracts meaningful concepts from the string of words. Since a conversation typically consists of alternating turns between speakers, usually the input module segments the user's speech into discrete quanta called *utterances* and produces a concept hypothesis for each utterance.

Accurate speech recognition is a difficult problem, and in practice reasonable accuracy can only be achieved by limiting the search to plausible sequences of words using a *language model*. Language models may be implemented either as *rule-based grammars* (such as context-free grammars), or *statistical language models* in the form of *N-grams* which estimate local probabilities of word sequences. Further gains are possible by considering a combined probability at the phoneme, word, language model, and parse levels when hypothesizing a parse for an utterance, or approximating this joint probability by using intermediate representations such as an *N-Best list* or a *lattice* structure [89]. The input module may also produce various recognition features, such as parse coverage, utterance length, indications of how well the acoustic data matched the models, etc. It is important to realize that even with state-of-the-art techniques, the input module will still make recognition errors.

The output module can also be thought of as a pipeline, in which a string of concepts are converted into an audio signal. The most general approach (and that shown in Figure 1.1) is to convert concepts into words using a *language generation* process and words into audio via *text to speech* (TTS). Alternatively, for more restricted domains, a direct mapping from concepts to pre-recorded audio fragments can be used.

The structure of the control module is rather different. The control module maintains a persistent state which tracks the machine's understanding of the conversation, a process called *dialog modelling*. As information is received from the input module, this state is updated. Based on this state, the machine decides what to say or do, a process called *dialog management*. The machine may take a communicative action (such as asking a question via the output module), a technical action (such as consulting a database, or changing a language model used by the input module), or some combination of these. Actions taken are also tracked in the persistent state, shown as the arrow from the dialog manager to the dialog model in Figure 1.1.

Whereas the input module, which classifies speech sounds into conceptual meaning, presents a reasonably clear optimization problem, it is much less clear how optimization should be performed in the control module. Different machine actions represent different trade-offs between speed, accuracy, user satisfaction, and other metrics: for example, taking time to verify the user's intentions can increase the accuracy of the control module's state, but also risks frustrating users

by prolonging the conversation. Making these trade-offs effectively is complicated because a conversation is a temporal process in which actions have both immediate and long-term effects, and because speech recognition errors ensure the control module never knows the true state of the dialog with certainty. As a result, development of the control module has traditionally been viewed as a user interface design problem in which iterative improvements are made by empirical observation of system performance. This process is labor intensive and expensive, and makes no guarantees that progress will be made toward an optimal control module.

More recently, researchers have begun applying optimization techniques to specific aspects of the control module. For example, machine learning techniques can consider the recognition features produced by the input module to produce a *confidence score* which gives an indication of the reliability of a single input hypothesis. A low confidence score indicates a recognition error is more likely, allowing the dialog manager to (for example) confirm its hypothesis with the user. Confidence scores are themselves not reliable and some erroneous information will inevitably be mistakenly accepted; as such it seems unwise to maintain just one hypothesis for the current dialog state. A more robust approach is to maintain *parallel state hypotheses* at each time-step. Finally, the consequences of mis-recognitions can be difficult for human designers to anticipate. Thus systems can perform *automated planning* to explore the effects of mis-recognitions and determine which sequence of actions are most useful in the long run.

These methods of coping with speech recognition errors – confidence scoring, automated planning, and parallel dialog hypotheses – can improve performance over handcrafted baselines, and confidence scores in particular are now routinely used in deployed systems. However, these methods typically focus on just a small part of the control module and rely on the use of *ad hoc* parameter setting (for example, hand-tuned parameter thresholds) and pre-programmed heuristics. Most seriously, they lack an overall statistical framework which can support global optimization.

This thesis argues that a partially observable Markov decision process (POMDP) provides a principled framework for control in spoken dialog systems. A POMDP is a mathematical model for machine planning distinguished by three properties. First, in a POMDP, the machine cannot detect with certainty the state of its world; rather it has sensors which provide only incomplete or potentially inaccurate information. Second, actions available to the machine have stochastic effects on the world, so the precise results of actions cannot be predicted. Finally, in a POMDP there exists a reward function which assigns real-valued “rewards” for taking certain actions in certain states, and the goal of the machine is to choose actions which maximize the *cumulative* sum of rewards gained over time.

In this thesis, the dialog management problem is cast as a POMDP to form a model called an SDS-POMDP (spoken dialog system partially observable Markov decision process). In the SDS-POMDP model, the state of the conversation is regarded as an unobserved variable, and rather than maintaining a single conversational state, a distribution over *all possible* conversational states is maintained and updated as new evidence is received. The machine is equipped with models of how users’ goals evolve, what actions users are likely to take, and how the speech

recognition process is likely to corrupt observations, which together enable the machine to update its distribution over time. The SDS-POMDP model enables these statistical models to be combined with domain knowledge in the form of rules specified by a (human) dialog designer. The dialog designer also provides objectives for the machine in the form of a reward function, and the machine chooses actions most likely to result in the highest discounted sum of these rewards over the course of the conversation.

Unfortunately, the expressive power of the POMDP approach is obtained at the expense of severe computational complexity. POMDPs are notoriously difficult to scale: in the dialog domain, as the size and complexity of the dialog model increases, the number of possible dialog states grows astronomically, and the planning process rapidly becomes intractable. Indeed, it will be shown that naively applying POMDP algorithms to real-world dialog management problems is hopelessly intractable. This thesis then shows how the SDS-POMDP model can be scaled to a real-world size by extending existing optimization algorithms to exploit characteristics of the SDS domain.

This thesis can be viewed in two halves. The first half, consisting of Chapters 2, 3, and 4, broadly addresses theory. Chapter 2 begins by reviewing POMDPs and relevant solution algorithms. Most of this chapter is devoted to explaining a POMDP optimization technique called “value iteration”, which will form the basis of techniques developed later on. Chapter 3 presents a detailed account of how a spoken dialog system can be cast as a POMDP to form the SDS-POMDP model, and how this model naturally reflects the major sources of uncertainty in spoken dialog systems. This chapter then describes a small but detailed SDS-POMDP called the TRAVEL application, and illustrates its optimization and operation. Chapter 4 then compares, at a theoretical level, the SDS-POMDP approach to related techniques in the literature and argues that the SDS-POMDP model subsumes and extends these existing techniques. Experimental results from dialog simulation on this TRAVEL application illustrate the benefits of the POMDP approach in quantitative terms.

The second half of this thesis tackles the problem of scaling the SDS-POMDP model to applications of a realistic size. Chapter 5 first limits the scope of consideration to so-called *slot-filling* dialogs, and explains the two main sources of exponential growth: the *number of slots*, and the *number of elements* in each slot. Chapter 5 then presents a method of optimization called “Summary point-based value iteration” (SPBVI) which addresses growth due to *number of elements* in each slot. A single-slot version of the TRAVEL application called MAXITRAVEL is presented and experimentation shows that the SPBVI method scales and outperforms common baselines. Next, Chapter 6 tackles scaling the *number of slots* by extending SPBVI to form “Composite summary point-based value iteration” (CSPBVI). An application called MAXITRAVEL-W is presented and experimentation using simulated dialogs again shows that CSPBVI scales well and outperforms common baselines. Finally, the construction and operation of a real spoken dialog system called TOURISTTICKETS is presented, which has been implemented using the SDS-POMDP model as a dialog manager optimized with CSPBVI. Chapter 7 concludes and suggests future lines of research.

POMDP Background

Partially observable Markov decision processes (POMDPs) are a mathematical framework for machine planning in an environment where actions have uncertain effects and where observations about the world provide incomplete or error-prone information. POMDPs have their origins in the 1960s Operations Research community [27, 2, 109, 107, 69], and have subsequently been adopted by the artificial intelligence community as a principled approach to planning under uncertainty [17, 51]. POMDPs serve as the core framework for the dialog model presented in this thesis.

This chapter first reviews the definition of POMDPs, both formally and through a very simple dialog management application. Next, exact POMDP optimization using “value iteration” is described, and applied to the example problem. Finally, since exact optimization is intractable for problems of a realistic size, an established technique for *approximate* optimization called “point-based value iteration” is reviewed, and applied to a larger version of the example dialog problem.

2.1 Introduction to POMDPs

Formally, a POMDP \mathfrak{P} is defined as a tuple $\mathfrak{P} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z}, \gamma, b_0)$ where \mathcal{S} is a set of states s describing the machine’s world with $s \in \mathcal{S}$; \mathcal{A} is a set of actions a that a machine may take $a \in \mathcal{A}$; \mathcal{T} defines a transition probability $P(s'|s, a)$; \mathcal{R} defines the expected (immediate, real-valued) reward $r(s, a) \in \mathfrak{R}$; \mathcal{O} is a set of observations o the machine can receive about the world with $o \in \mathcal{O}$; \mathcal{Z} defines an observation probability $P(o'|s', a)$; γ is a geometric discount factor $0 < \gamma < 1$; and b_0 is an initial belief state, defined below.

The POMDP operates as follows. At each time-step, the world is in some unobserved state s . Since s is not known exactly, a distribution over states is maintained called a *belief state*, b . b is

defined in *belief space* \mathcal{B} , which is an $(|\mathcal{S}| - 1)$ -dimensional simplex:

$$\begin{aligned} \mathcal{B} = \{(b(s_1), \dots, b(s_{|\mathcal{S}|}))\} : & \quad b(s) \in \mathfrak{R}, \forall s \in \mathcal{S}, \text{ and} \\ & \quad \sum_s b(s) = 1, \text{ and} \\ & \quad b(s) \geq 0, \forall s \in \mathcal{S}. \end{aligned}$$

Based on b , the machine selects an action a , receives a reward $r(s, a)$, and transitions to (unobserved) state s' , where s' depends only on s and a . The machine then receives an observation o' which depends on s' and a . At each time-step, $b(s)$ can be updated using a function called the *state estimator (SE)*, which computes a new distribution over states b' given a current distribution over states b , an action taken a , and an observation received o' :

$$SE(b, a, o') = b'(s') \tag{2.1}$$

$$\begin{aligned} &= P(s'|o', a, b) \\ &= \frac{P(o'|s', a, b)P(s'|a, b)}{P(o'|a, b)} \\ &= \frac{P(o'|s', a) \sum_s P(s'|a, b, s)P(s|a, b)}{P(o'|a, b)} \\ &= \frac{P(o'|s', a) \sum_s P(s'|a, s)b(s)}{P(o'|a, b)}. \end{aligned} \tag{2.2}$$

The numerator consists of the observation function \mathcal{Z} , transition matrix \mathcal{T} , and current belief state b . The denominator is independent of s' , and can be regarded as a normalization constant η ; hence:

$$b'(s') = \eta \cdot P(o'|s', a) \sum_s P(s'|a, s)b(s) \tag{2.3}$$

The process of maintaining at each time-step τ the belief state b_τ is called “belief monitoring” and is written

$$b_{\tau+1} \leftarrow SE(b_\tau, a_\tau, o_{\tau+1}) \tag{2.4}$$

where b_τ gives the belief state at time τ , a_τ gives the action taken at time τ , and $o_{\tau+1}$ gives the observation received as a result of taking a_τ . An important property of the belief state is that it is a *sufficient statistic* which summarizes the initial belief state b_0 , all of the actions taken, and all of the observations received. Stated more formally:

$$b_\tau(s) = p(s|b_0, a_0, o_1, a_1, o_2, \dots, a_{\tau-1}, o_\tau). \tag{2.5}$$

At each time-step τ , the machine receives reward r_τ which (as mentioned above) depends on the current state and action, $r(s, a)$. The cumulative, discounted reward accumulated by time-step t is written V_t and is computed as

$$V_t = \sum_{\tau=0}^{t-1} \gamma^\tau r_\tau \tag{2.6}$$

The cumulative, discounted, *infinite horizon* reward is called the *return*, denoted by V_∞ , or simply V for short. The goal of the machine is to choose actions in such a way as to maximize the expected return $E[V]$ given the POMDP parameters $\mathfrak{P} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z}, \gamma, b_0)$. Methods for achieving this are the focus of the rest of this chapter, beginning at section 2.2.

Example VOICEMAIL POMDP

To illustrate the basic operation of POMDPs and to introduce how POMDPs will be used in a spoken dialog system, an example will now be presented in some detail. This example concerns a very simple application called VOICEMAIL which although very limited, nevertheless demonstrates the essential properties of a POMDP in the SDS domain.¹ The remainder of the thesis extends the POMDP model to more sophisticated applications.

In this example, users listen to voicemail messages and at the end of each message, they can either *save* or *delete* the message. We refer to these as the user’s goals and since the system does not *a priori* know which goal the user desires, they are hidden goals and form the unobservable state: $\mathcal{S} = \{\text{save}, \text{delete}\}$. For the duration of the interaction relating to each message, the user’s goal is fixed, and the POMDP-based dialog manager is trying to guess which goal the user has.

Belief space has two elements, $b(s = \text{save})$ and $b(s = \text{delete})$, and each point in this space b represents a distribution over these two goals. Figure 2.1 shows a graphical depiction of belief space. Since there are only two states, belief space can be depicted as a line segment. In this depiction, the ends of the segment (in general called *corners*) represent certainty. For example, $b = (1, 0)$, the left end of the line segment, indicates $b(s = \text{save}) = 1$ and $b(s = \text{delete}) = 0$; in other words, $s = \text{save}$ with certainty. Similarly, $b = (0, 1)$, the right end of the line segment, indicates certainty that $s = \text{delete}$, and intermediate points represent varying degrees of certainty in the user’s goal. The one belief point shown is the initial belief state, $b_0 = (0.65, 0.35)$, which indicates that the user’s goal is more likely to be *save* than *delete*.

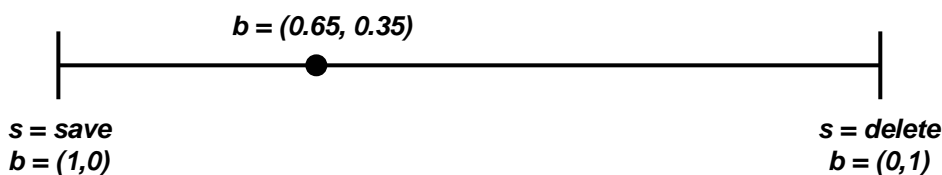


Figure 2.1 *Belief space in a POMDP with two states, save and delete, which correspond to hidden user goals. At each time-step, the current belief state is a point on this line segment. The ends of the line segment represent certainty in the current state. The belief point shown is the initial belief state.*

The machine has only three available actions: it can *ask* what the user wishes to do in order to infer his or her current goal, or it can *doSave* or *doDelete* and move to the next message. When the user responds to a question, it is decoded as either the observation $\overline{\text{save}}$ or $\overline{\text{delete}}$. However, since speech recognition errors can corrupt the user’s response, these observations cannot be

¹Readers may recognize this POMDP as a variation of the well-known “Tiger” problem cast into the spoken dialog domain [17].

used to deduce the user’s intent with certainty. If the user says “save” then an error may occur with probability 0.2, whereas if the user says “delete” then an error may occur with probability 0.3. After a *doSave* or *doDelete* action, the machine moves on to the next message and the user selects a (possibly) new goal, returning the belief state to its original value via the transition function. After a *doSave* or *doDelete* action, there is no information from the speech recognizer, and this is expressed in the POMDP by removing the conditioning of the observation on actions and states by setting $P(o'|a, s') = P(o')$.²

The designer of this system specifies the objectives of the machine via a reward function. The machine receives a large positive reward (+5) for getting the user’s goal correct, a very large negative reward (−20) for taking the action *doDelete* when the user wanted *save* (since the user may have lost important information), and a smaller but still significant negative reward (−10) for taking the action *doSave* when the user wanted *delete* (since the user can always delete the message later). There is also a small negative reward for taking the *ask* action (−1), since (all else being equal) the machine should try to process messages as quickly as possible. The transition dynamics of the system are shown in Tables 2.1, 2.2, and 2.3. The discount factor γ in this example is set to $\gamma = 0.95$.

a	s	$s' = \text{save}$	$s' = \text{delete}$
<i>ask</i>	<i>save</i>	1.00	0.00
	<i>delete</i>	0.00	1.00
<i>doSave</i>	<i>save</i>	0.65	0.35
	<i>delete</i>	0.65	0.35
<i>doDelete</i>	<i>save</i>	0.65	0.35
	<i>delete</i>	0.65	0.35

Table 2.1 Transition function $P(s'|s, a)$ for the example VOICEMAIL spoken dialog system POMDP. The state s indicates the user’s goal as each new voicemail message is encountered.

a	s'	$o' = \overline{\text{save}}$	$o' = \overline{\text{delete}}$
<i>ask</i>	<i>save</i>	0.80	0.20
	<i>delete</i>	0.30	0.70
<i>doSave</i>	<i>save</i>	0.50	0.50
	<i>delete</i>	0.50	0.50
<i>doDelete</i>	<i>save</i>	0.50	0.50
	<i>delete</i>	0.50	0.50

Table 2.2 Observation function $P(o'|a, s')$ for the example VOICEMAIL spoken dialog system POMDP. Note that the observation o' only conveys useful information following an *ask* action.

As the machine takes actions and receives observations, it performs belief monitoring to better estimate the current state. Figure 2.2 illustrates this process, in which the machine takes

²Here $P(o')$ has been arbitrarily set to 0.5.

a	$s = \text{save}$	$s = \text{delete}$
ask	-1	-1
doSave	+5	-10
doDelete	-20	+5

Table 2.3 Reward function $r(s, a)$ for the example VOICEMAIL spoken dialog system POMDP. The values encode the dialog design criteria where it is assumed that deleting wanted messages should carry a higher penalty than saving unwanted messages, and where wasting time by repeatedly asking questions should be discouraged.

actions *randomly*. (Below techniques for choosing actions in order to maximize return will be considered.) The POMDP starts at time-step $t = 0$ in initial belief state $b_0 = (0.65, 0.35)$, and a random unobserved state which is in this illustration $s_0 = \text{save}$. At each time-step, the machine takes an action a , receives a reward r , advances to the next time-step, transitions to a (possibly) new hidden state s' , and receives an observation o' .

In this illustration, at $t = 0$, the system takes the action ask and receives reward -1 (as given in Table 2.3). The machine then advances to time-step $t = 1$. The next (hidden) state is chosen according to the transition function $P(s'|s, a)$, and the transition function for this POMDP given in Table 2.1 ensures the user's goal remains constant after an ask machine action. The machine then receives an observation according to the observation function $P(o'|s', a)$ given in Table 2.2, and in this illustration a speech recognition error is made which causes the user's response of "save" to be corrupted and observed by the system as $\overline{\text{delete}}$. The initial belief state b_0 , action a and observation o and used to compute a new belief state b_1 using Equation 2.3. Note that the belief state has moved toward the $s = \text{delete}$ corner, reflecting the evidence received. In $t = 1$, the machine then (randomly) takes the action doDelete . The reward -20 is received, time advances to $t = 2$, a (possibly) new state is sampled from the transition function, an observation is sampled from the observation function, and so on.

In this example, the total discounted reward accumulated at $t = 1$ (calculated using Equation 2.6) is $V_2 = \gamma^0 \cdot -1 + \gamma^1 \cdot -20 = -1 + 0.95 \cdot -20 = -20$. Taking actions at random is of course unlikely to yield optimal results, and the rest of this chapter discusses techniques for maximizing returns.

2.2 Finding POMDP policies

Maximizing V in practice means finding a plan called a *policy* which indicates which actions to take at each turn. POMDP policies can take on various forms, including a collection of *conditional plans* or a *partitioning* of belief space.³ These forms are related, and each will be described, starting with a conditional plan.

³A finite state machine (FSM) is another form. In this thesis, FSMs will not be used for optimization, but will be used to evaluate handcrafted dialog managers, in section 4.4, page 58.

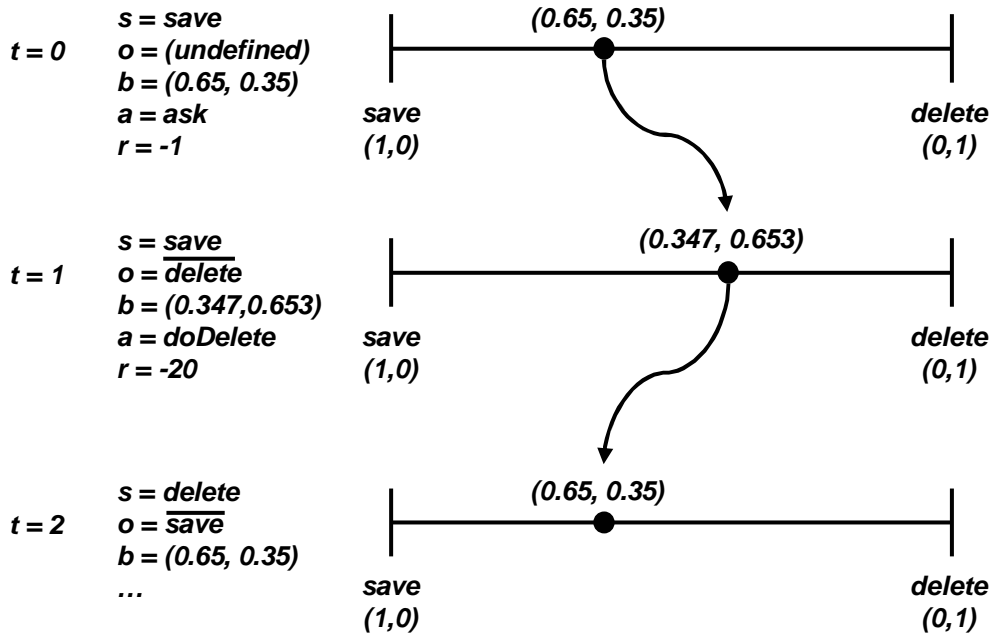


Figure 2.2 Illustration of the belief monitoring process in the VOICEMAIL spoken dialog system POMDP. Updates to the belief state are computed using Equation 2.3. The initial state (i.e., the initial user goal) is save. After the system takes the doDelete action, the next user goal is delete.

A t -step conditional plan describes a policy with a horizon of t steps into the future. Formally, a t -step conditional plan is a tree of uniform depth t and constant branching factor $|\mathcal{O}|$, in which each node is labelled with an action. The root node is referred to as layer t and the leaf nodes are referred to as layer 1. Every non-leaf node has $|\mathcal{O}|$ children, indexed as $1, 2, \dots, |\mathcal{O}|$. A conditional plan is used to choose actions by first taking the action specified by the root node (layer t). An observation o will then be received from the POMDP, and control passes along arc o to a node in layer $t - 1$. The action specified by that node is taken, and so on. In this way, a t -step conditional plan specifies a policy which extends t steps into the future.

As an illustration, two example conditional plans for the VOICEMAIL spoken dialog system POMDP are shown in Figure 2.3. Conditional plan A first takes the *ask* action then takes the corresponding “do” action immediately, followed by another *ask* action. Conditional plan B is more conservative, and only takes a “do” action if it receives two consistent observations; otherwise, it takes *ask* actions.

A t -step conditional plan has a value $V(s)$ associated with it, which indicates the expected value of the conditional plan, depending on the current (unobserved) state s . This value can be calculated recursively for $\tau = 0 \dots t$ as:

$$V_\tau(s) = \begin{cases} 0, & \text{if } \tau = 0, \\ r(s, a_\tau) + \gamma \sum_{s'} P(s'|s, a_\tau) \sum_{o'} P(o'|s', a_\tau) V_{\tau-1}^{o'}(s'), & \text{otherwise} \end{cases} \quad (2.7)$$

where a_τ gives the action associated with the root node of this conditional plan, and $V_{\tau-1}^{o'}$ indicates the value of the conditional plan in layer $\tau - 1$ which is the child index o' of the conditional plan.

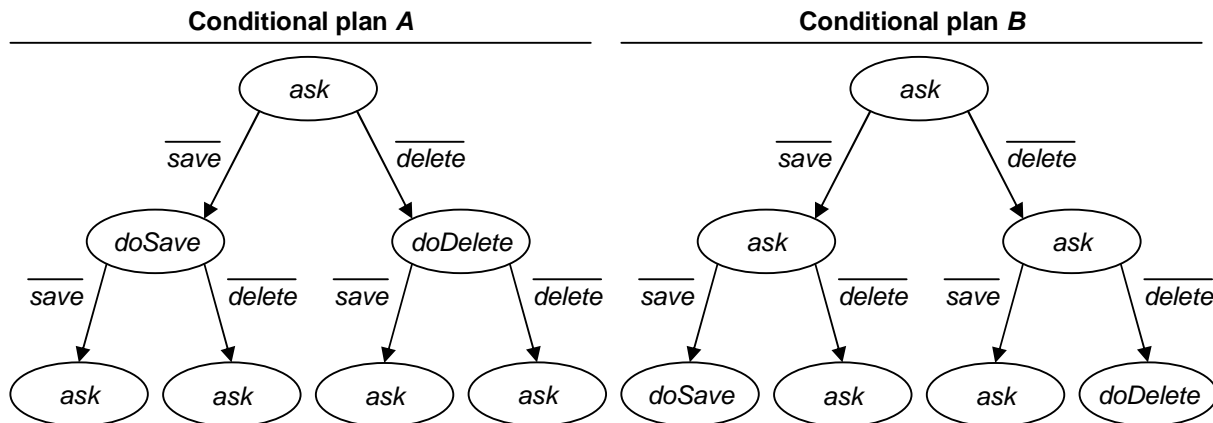


Figure 2.3 Two example 3-step conditional plans for the example VOICEMAIL POMDP application. In this POMDP, $|\mathcal{O}| = 2$ and $|\mathcal{A}| = 3$. Note that each non-leaf node has exactly $|\mathcal{O}|$ children, labelled $\overline{\text{delete}}$ and $\overline{\text{save}}$, and each node is labelled with an action.

As an illustration, the values of the two conditional plans shown in Figure 2.3 are now calculated by applying Equation 2.7 repeatedly. Figure 2.4 depicts this process for the two example conditional plans. The notation $[-20, +5]$ indicates that the value of a conditional plan is $V(s = \text{save}) = -20$ and $V(s = \text{delete}) = +5$. The value of conditional plan A is $[-1.9025, -1.4275]$ and the value of conditional plan B is $[-1.2147, -0.2576]$.

At runtime, the machine doesn't know the state s exactly and rather maintains a belief state b , so for a machine to evaluate a conditional plan, a definition of $V(b)$ is needed. The value of a conditional plan at a belief state B is computed as an expectation over states:

$$V(b) = \sum_s b(s)V(s). \tag{2.8}$$

An expectation can be taken because b is a complete summary of all of the actions and observations up to the current time-step in the dialog. More formally, for a given initial belief state b_0 and history $(a_1, o_1, a_2, o_2, \dots, a_n, o_n)$, b provides a proper sufficient statistic: b is Markovian with respect to b_0 and $(a_1, o_1, a_2, o_2, \dots, a_n, o_n)$ [51].⁴

As an illustration, $V(b)$ is shown for the two example conditional plans in Figure 2.5. As in Figure 2.1, the horizontal axis represents belief space. The vertical axis now represents the *value* of a conditional plan as a function of belief state. Since the value of a conditional plan at a belief state is an expectation over states (Equation 2.8), the value of each conditional plan is a line segment in this graph. If these two conditional plans form the set \mathcal{N}_3 , then the upper surface of these two line segments represents $V_{\mathcal{N}_3}^*(b)$ – value of choosing the optimal 3-step policy in \mathcal{N} .

In a POMDP, the machine's task is to choose between a number of conditional plans to find the one which maximizes V_t . Given a set of t -step conditional plans \mathcal{N}_t with $n \in \mathcal{N}_t$, and their

⁴This is a statement of mathematical theory, and it is a separate, open question whether user state and behavior can be accurately captured in a Markovian model, since the true internal state representation of the user is of course unknown. Nevertheless, assuming that a user's state and behavior can be expressed in a Markovian model is a useful approximation since it allows plans to be constructed without exhaustively enumerating all possible dialogs.

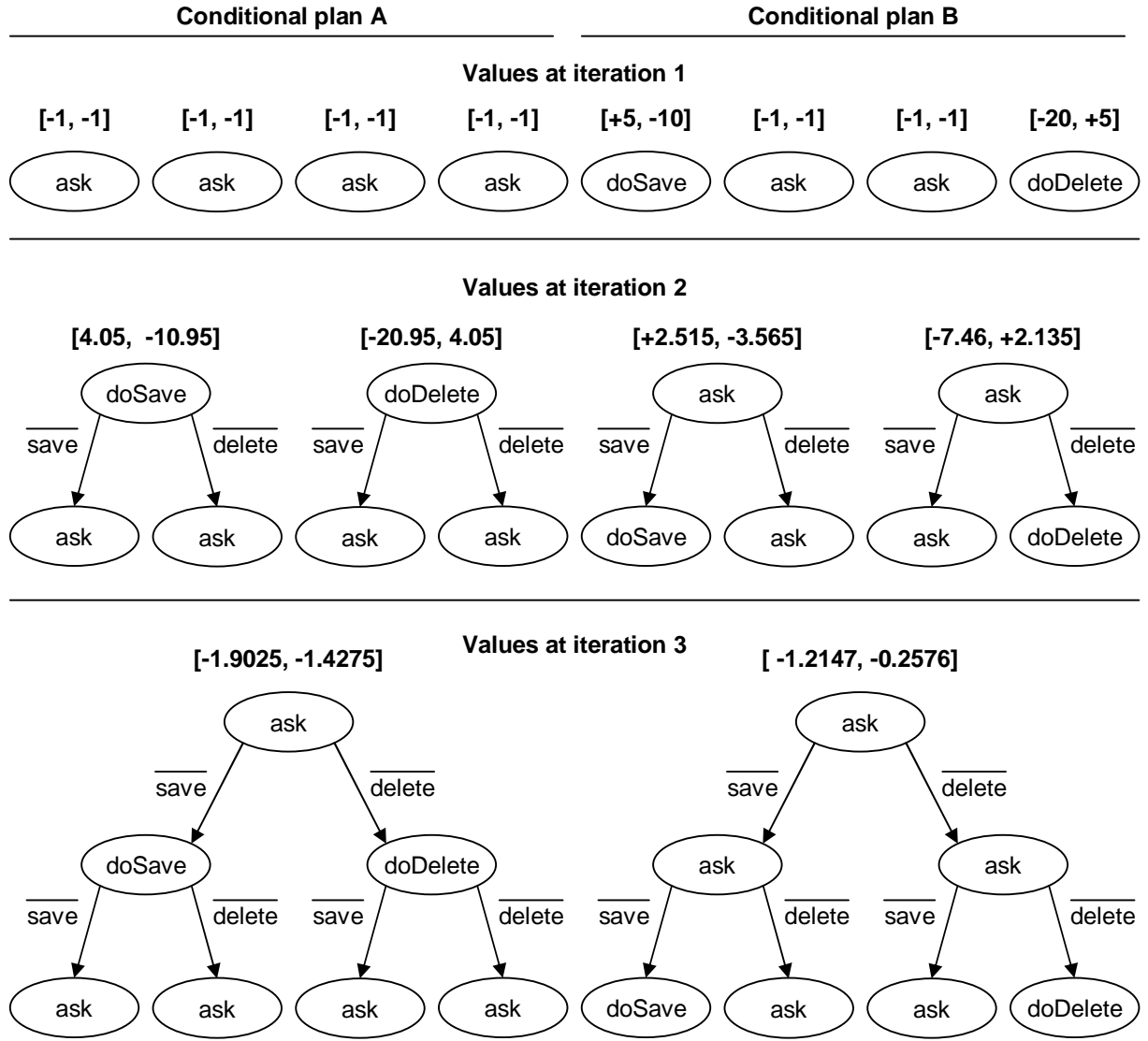


Figure 2.4 Calculation of a value function for the 3-step conditional plans shown in Figure 2.3, computed by repeatedly applying Equation 2.7.

corresponding values $\{V_t^n\}$ and initial actions $\{a_t^n\}$, the value of the best plan at belief state b is:

$$V_{\mathcal{N}_t}^*(b) = \max_n \sum_s b(s) V_t^n(s). \tag{2.9}$$

$V_{\mathcal{N}_t}^*(b)$ implies an *optimal policy* $\pi_{\mathcal{N}_t}^*(b)$:

$$\pi_{\mathcal{N}_t}^*(b) = a_t^n \text{ where } n = \arg \max_n \sum_s b(s) V_t^n(s). \tag{2.10}$$

In words, $V_{\mathcal{N}_t}^*(b)$ represents the (scalar) expected value of starting in b and following the best t -step conditional plan in \mathcal{N}_t , which begins with action $\pi_{\mathcal{N}_t}^*(b)$.

In the illustration in Figure 2.5, the upper surface is always conditional plan B, which indicates that conditional plan B yields a higher expected return at any belief state. This seems

intuitive because, in the face of large negative rewards for saving or deleting messages erroneously, conditional plan B proceeds more cautiously than conditional plan A , gathering more evidence before taking a “do” action.

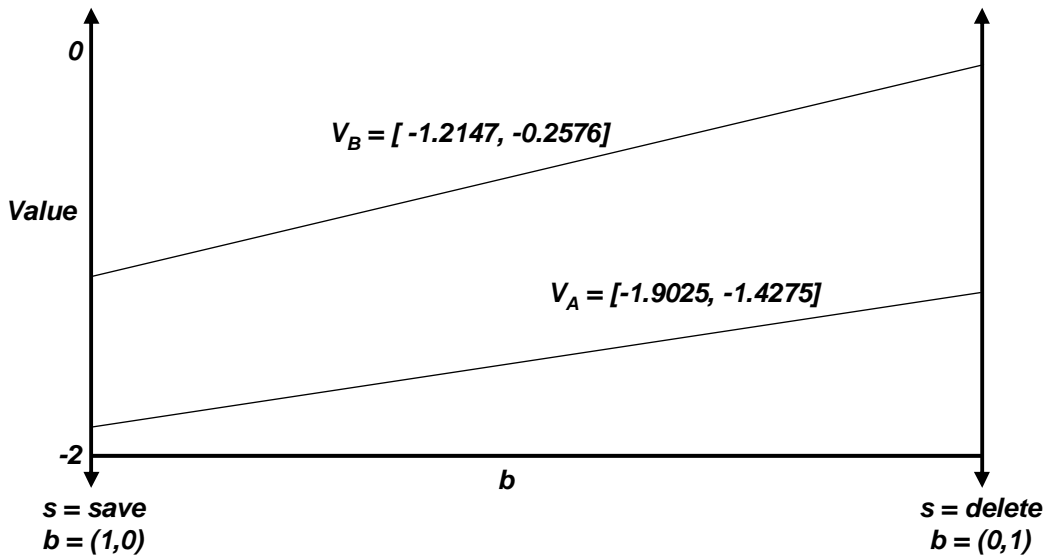


Figure 2.5 Example value function for the two conditional plans shown in Figure 2.3. Note that conditional plan B yields a higher expected value for any belief state b .

The structure in Equation 2.9 leads to an important insight. The value of each individual conditional plan, $V_t^n(b)$, is an expectation over states – i.e., a hyperplane in belief space. Since the optimal policy takes a *max* over many hyperplanes, this causes the value function of an optimal policy $V_{\mathcal{N}_t}^*(b)$ to be *piece-wise linear* and *convex*, and the optimal value function is formed of regions where one hyperplane (i.e., one conditional plan) is optimal [109, 107].

If \mathcal{N}_t contains all possible conditional plans, then $V_{\mathcal{N}_t}^*(b)$ gives the value of the *optimal t -step policy* for this POMDP, written V_t^* , where $\pi_t^*(b) \in \mathcal{A}$ indicates the first action of the best conditional plan. As t approaches infinity, the value function converges, to V_∞^* or simply V^* . In practice of course an optimization algorithm can’t be run infinitely long and thus V^* cannot be computed directly, but fortunately it is the case that V^* can be approximated arbitrarily closely with \bar{V}_t^* by considering a large enough t , where \bar{V}_t^* is the value obtained by at every time-step choosing actions using $\pi_t^*(b)$ – i.e., following V_t^* as if there are always t time-steps to go.

More formally, the largest difference between V_{t-1}^* and V_{t-2}^* is referred to as the *Bellman residual*,

$$\text{Bellman residual} = \sup_b |V_{t-1}^*(b) - V_{t-2}^*(b)|. \quad (2.11)$$

If the Bellman residual is bounded by some δ such that

$$\sup_b |V_{t-1}^*(b) - V_{t-2}^*(b)| \leq \delta, \quad (2.12)$$

then it can be shown that \bar{V}_t^* differs from V^* by at most $2\delta\gamma/(1-\gamma)$ [88]. If δ is chosen to be

$\varepsilon(1 - \gamma)/2\gamma$, then a bound can be stated on the quality of the approximation \bar{V}_t^* :

$$\sup_b |\bar{V}_t^*(b) - V^*(b)| \leq \varepsilon. \quad (2.13)$$

Finally, it can be also be shown that δ decreases by a factor of at least γ with each increment in t [88], so in practice δ (and by extension ε) can be made arbitrarily small, and a finite-horizon value function \bar{V}_t^* may be computed which is arbitrarily close to V^* . Once an acceptable V_t^* has been calculated, it can be used to generate a policy using Equation 2.10.

The above analysis suggests that a simple, brute-force method of finding a nearly-optimal policy would be to choose a large value of t and enumerate all possible t -step conditional plans, \mathcal{N}_t . Unfortunately, computing optimal policies for even small values of t in this way is hopelessly intractable, because the number of possible conditional plans grows astronomically in t . In fact, the number of possible t -step conditional plans is

$$|\mathcal{A}|^{\frac{|\mathcal{O}|^t - 1}{|\mathcal{O}| - 1}}. \quad (2.14)$$

In words, a t -step policy tree contains $(|\mathcal{O}|^t - 1)/(|\mathcal{O}| - 1)$ nodes, and each node can be labelled with one of $|\mathcal{A}|$ possible actions [17]. To find the optimal t -step policy, all conceivable t -step conditional plans would need to be enumerated and evaluated, which is intractable for all but the most trivial POMDPs.

In the trivial VOICEMAIL example, there are a total of 2,187 3-step conditional plans, rising to 10^7 4-step conditional plans, 10^{15} 5-step conditional plans, and 10^{30} 6-step conditional plans. Clearly complete enumeration is hopeless, but this example suggests the possibility of an *iterative, incremental* approach: note that whether or not conditional plan A is included in \mathcal{N}_3 , $V_{\mathcal{N}_3}^*(b)$ is the same because the \max operation finds the upper surface of all of the value functions, and nowhere is conditional plan A optimal (Figure 2.5). When building conditional plans with longer horizons, A will never be included as a (3-step) child of an optimal 4-step conditional plan and can safely be discarded. Empirically it has been found that many vectors are fully dominated by one (or more) other vectors in most POMDPs, and as a result usually only a small number of conditional plans usually make a contribution to the optimal policy. This notion is the intuition behind value iteration, described next.⁵

2.3 Value iteration

Empirically, it has been found that relatively few t -step conditional plans make a contribution to an optimal t -step policy, and this insight can be exploited to compute optimal policies more efficiently with *value iteration* [69, 109]. Value iteration is an exact, iterative, dynamic programming process in which successively longer planning horizons are considered, and an optimal policy is incrementally created for longer and longer horizons. Value iteration proceeds by finding the *subset* of possible t -step conditional plans which contribute to the optimal t -step policy.

⁵The techniques in this thesis are based on value iteration; for a review of other techniques see [74].

These conditional plans are called *useful*, and only useful t -step plans are considered when finding the $(t + 1)$ -step optimal policy. The value iteration algorithm for POMDPs is shown in Algorithm 1 [69, 51].

Algorithm 1: Value iteration.

```

Input:  $\mathfrak{B}, T$ 
Output:  $\{V_T^n\}, \{a_T^n\}$ 
1 foreach  $s \in \mathcal{S}$  do
2    $V_0(s) \leftarrow 0$ 
3  $N \leftarrow 1$  //  $N$  is the number of  $(t - 1)$ -step conditional plans.
4 for  $t \leftarrow 1$  to  $T$  do
5   // Generate  $\{v^{a,k}\}$ , values of all possibly useful CPs.
    $\mathcal{K} \leftarrow \{V_{t-1}^n : 1 \leq n \leq N\}^{|\mathcal{O}|}$ 
   //  $\mathcal{K}$  now contains  $N^{|\mathcal{O}|}$  elements, where each
   // element  $k$  is a vector  $k = (V_{t-1}^{x_1}, \dots, V_{t-1}^{x_{|\mathcal{O}|}})$ .
   // This growth is the source of the computational complexity.
6   foreach  $a \in \mathcal{A}$  do
7     foreach  $k \in \mathcal{K}$  do
8       foreach  $s \in \mathcal{S}$  do
9         // Notation  $k(o')$  refers to element  $o'$  of vector  $k$ .
          $v^{a,k}(s) \leftarrow r(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_{o'} P(o'|s', a) V_{t-1}^{k(o')}(s')$ 
10        // Prune  $\{v^{a,k}\}$  to yield  $\{V_t^n\}$ , values of actually useful CPs.
        //  $n$  is the number of  $t$ -step conditional plans.
         $n \leftarrow 0$ 
11       foreach  $a \in \mathcal{A}$  do
12         foreach  $k \in \mathcal{K}$  do
13           // If the value of plan  $v^{a,k}$  is optimal anywhere in  $\mathcal{B}$ ,
           // it is ‘useful’ and will be kept.
           if  $\exists b : v^{a,k}(b) = \max_{\tilde{a}, \tilde{k}} v^{\tilde{a}, \tilde{k}}(b)$  then
14              $n \leftarrow n + 1$ 
15              $a_t^n \leftarrow a$ 
16             foreach  $s \in \mathcal{S}$  do
17                $V_t^n(s) \leftarrow v^{a,k}(s)$ 
18            $N \leftarrow n$ 

```

Each iteration of Algorithm 1 contains two steps. First, in the “generation” step, all potentially useful t -step conditional plans are created by enumerating all actions followed by all possible useful combinations of $(t - 1)$ -step plans. Then, in the “pruning” step, conditional plans

which do not contribute to the optimal t -step policy are removed, leaving the set of useful t -step plans. The algorithm is repeated for T steps.⁶

Although $\{V_t^n\}$ refers to the values of t -step policy *trees*, in practice only the root node’s action a_t^n of any tree will ever be taken when the policy is executed, so in value iteration, only one action needs to be stored for each conditional plan. It may seem odd that value iteration finds the value of optimal conditional plans, but disregards all of the actual content of each plan, except for the first action. In effect, when a policy produced by running T steps of value iteration is executed, it will be assumed that T is sufficiently close to infinity, and thus that there are always T time-steps to go – i.e., at runtime, a (possibly) different policy tree is selected at each time-step and *its* action is taken.

Example of value iteration in the VOICEMAIL application

Figure 2.6 shows the first step of value iteration applied to the VOICEMAIL spoken dialog POMDP example problem. In the first step, there are 3 possible conditional plans, one for each action. This figure shows the three 1-step conditional plan values $\{V_1^1(b), V_1^2(b), V_1^3(b)\}$, and the heavy line shows V_1^* , the value of the optimal 1-step policy.

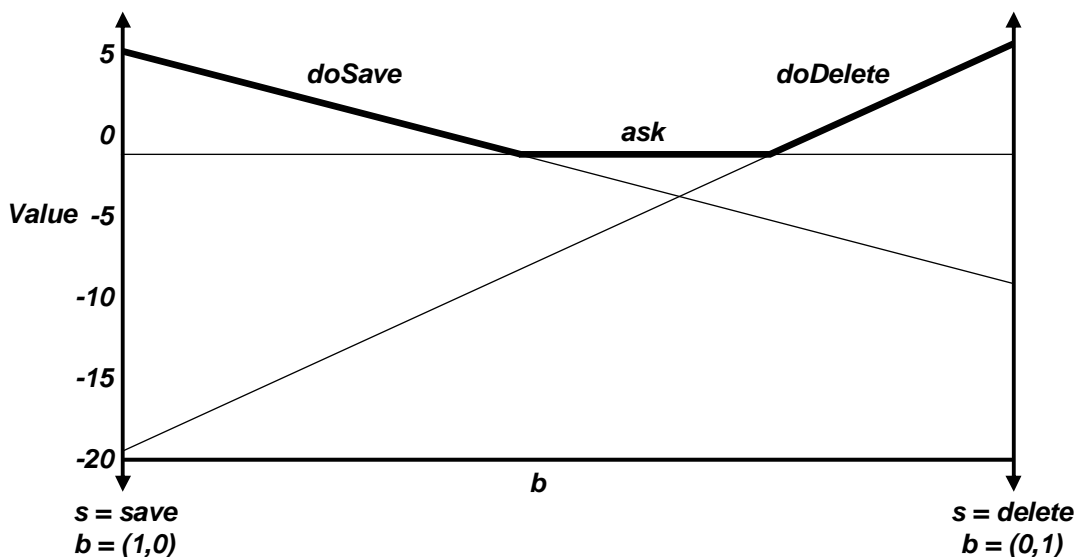


Figure 2.6 First step of value iteration on the VOICEMAIL example POMDP. Conditional plans are labelled with their initial actions. The heavy line shows $V_1^*(b)$, the value of the optimal 1-step policy.

The generation phase of the second step of value iteration produces 27 potentially useful 2-step conditional plans, shown in Figure 2.7. In the pruning step, plans which don’t contribute to the optimal policy are pruned, and in this example 22 of the 27 conditional plans are pruned,

⁶In the literature, typically value iteration is run until a terminating condition $\sup_b |V_t^*(b) - V_{t-1}^*(b)| < \epsilon$ is reached [51]. In this thesis appropriate numbers of iterations will be found empirically – for example, in the next chapter Figure 3.5, page 38. In any case, dialogs are episodic tasks for which a maximum length can be posited, and so using a constant but large horizon will be sufficient.

leaving 5 plans which contribute to the optimal policy. Figure 2.8 shows the values of those 2-step conditional plans which contribute to the optimal policy. The heavy line shows $V_2^*(b)$.

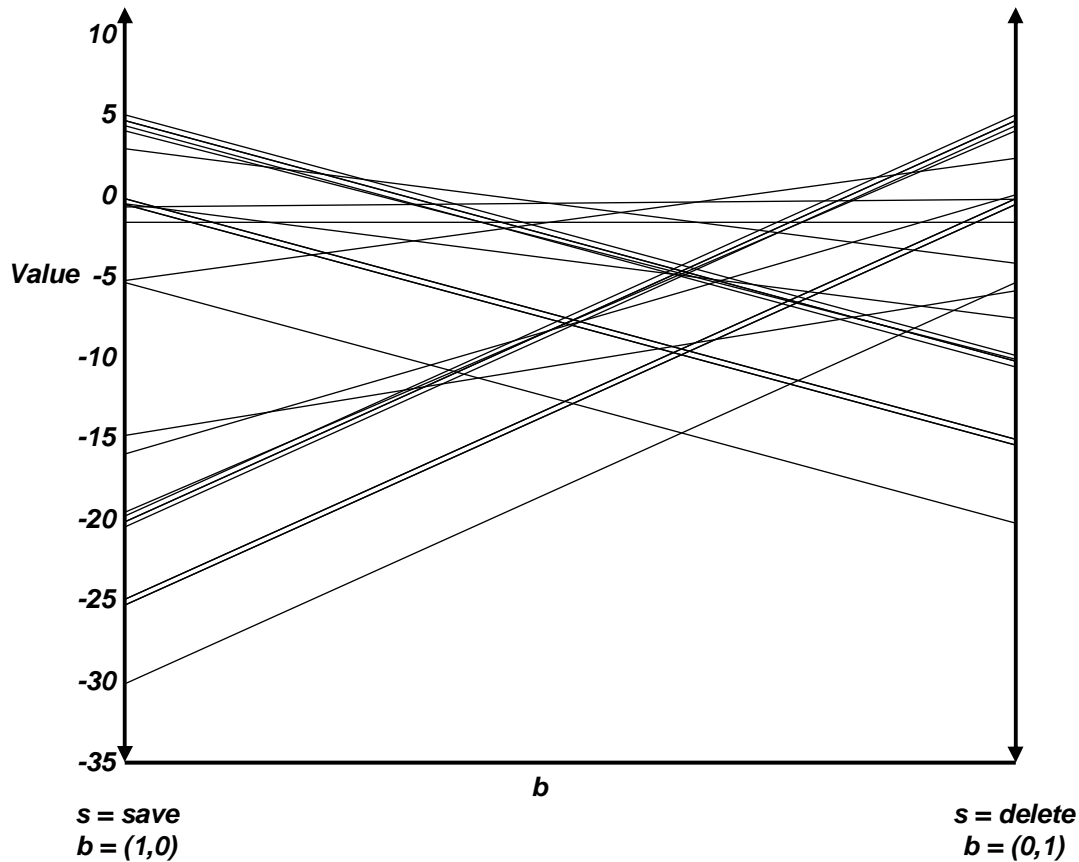


Figure 2.7 Second step of value iteration on the VOICEMAIL example POMDP before pruning, showing the values of all 27 potentially useful 2-step conditional plans.

Value iteration continues in this way for 100 iterations, and the resulting value function containing 34 vectors is shown in Figure 2.9.⁷ The upper surface of these 34 vectors represents $V^*(b)$, the value of the optimal infinite-horizon policy. The leftmost vector gives the value of a conditional plan which starts with the *doSave* action; the rightmost vector gives the value of a conditional plan which starts with the *doDelete* action, and all of the other vectors give the value of conditional plans which start with the *ask* action. The optimal policy $\pi^*(b)$ is shown in Figure 2.9 and consists of three *partitions* where a single action is optimal (these partitions are delineated with dashed lines in Figure 2.9.) In the regions of belief space close to the corners (where certainty is high), the machine chooses *doSave* or *doDelete*; in the middle of belief space (where certainty is low) it chooses to gather information with the *ask* action. Further, since the penalty for wrongly choosing *doDelete* is worse than for wrongly choosing *doSave*, the *doDelete* region is smaller: the machine requires more certainty to take the *doDelete* action. This policy (i.e., partitioning) is optimal in that no other partitioning, when averaged over many iterations, will achieve a higher return.

⁷100 iterations has been selected as it is assumed that no dialog will exceed 100 turns.

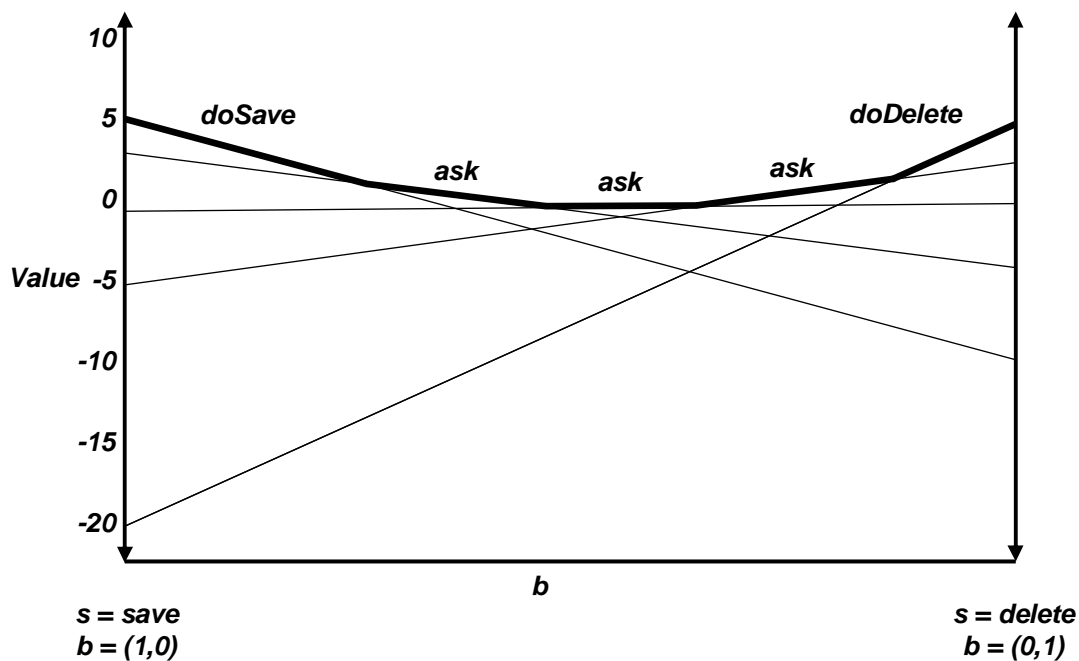


Figure 2.8 Second step of value iteration on the VOICEMAIL example POMDP after pruning, showing the values of the 5 conditional plans which contribute to the optimal 2-step policy. Conditional plans are labelled with their initial actions. The heavy line shows $V_2^*(b)$, the value of the optimal 2-step policy.

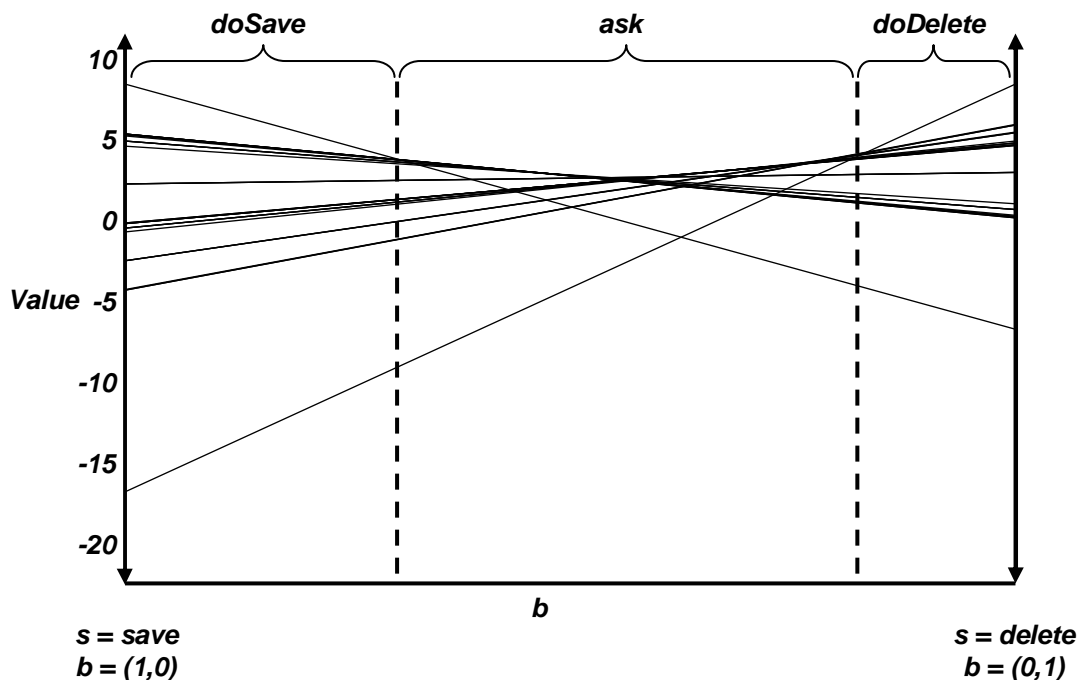


Figure 2.9 Terminal step of value iteration in the VOICEMAIL example POMDP. The upper surface of the value function shows $V^*(b)$, the optimal infinite-horizon policy. The dashed lines show portions of belief space where one action is optimal.

Figure 2.10 shows an example conversation between a user and a machine executing the optimal policy. At each time-step, the machine action and the observation are used to update the belief state as in Equation 2.3. Actions are selected depending on the partition which contains the current belief state. In this example, the first response is mis-recognized moving the belief state towards the *delete* corner. However, since the belief state remains in the central region where uncertainty is high, the machine continues to *ask* the user what to do. After two successive *save* observations, the belief state moves into the *doSave* region, the message is saved and the belief state transitions back to the initial belief state b_0 . The return for processing this message is +6.6212.

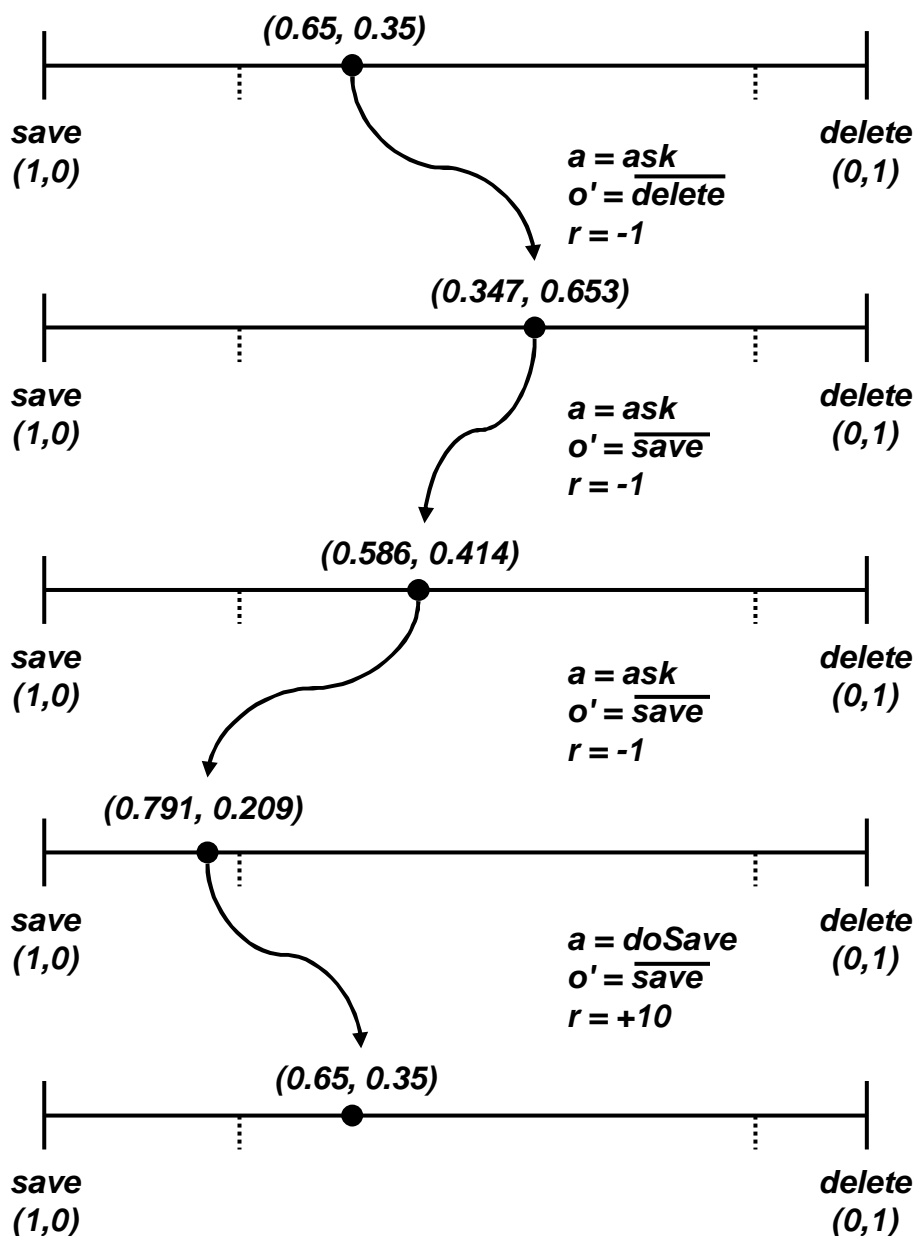


Figure 2.10 Example conversation with a machine executing the optimal policy for the VOICEMAIL example spoken dialog POMDP. In this example the user's goal is save. The dashed lines show the partition policy, given in Figure 2.9. Note that a recognition error is made after the first ask action.

Value iteration can be significantly more efficient than exhaustive enumeration. At each iteration of value iteration, $\{v^{a,k}\}$ contains $|\mathcal{A}||\mathcal{V}_{t-1}|^{|\mathcal{O}|}$ conditional plans (line 6 in Algorithm 1 iterates over $|\mathcal{A}|$ actions, then for each action line 7 iterates over $|\mathcal{V}_{t-1}|^{|\mathcal{O}|}$ combinations of successor plans), and improvements to basic POMDP value iteration such as the *Witness* algorithm avoid generating all elements of $\{v^{a,k}\}$, decreasing the number of conditional plans which must be considered at each iteration [51]. Even so, in practice, the combination of growth in the number of conditional plans, and the computational complexity of the “pruning” operation cause exact value iteration to be intractable for problems on the order of 10 states, actions, and observations.⁸ To scale to problems of a realistic size, *approximations* must be made to the exact optimal policy, and these are discussed next.

2.4 Point-based value iteration

Value iteration is computationally complex primarily because it attempts to find an optimal policy for *all points* in belief space \mathcal{B} . As a result the generation step can produce many more vectors than can possibly be analyzed, because the search for useful vectors (the “prune” step in Algorithm 1) searches continuously-valued belief space. *Point-based value iteration* (PBVI) [85], by contrast, finds optimal conditional plans *only* at a finite set of N discrete belief points in belief space, $\hat{\mathcal{B}} = \{b_1, b_2, \dots, b_N\}$. The value of each of these conditional plans, $V_t^n(s)$, is *exact*, but only guaranteed to be *optimal* at b_n , and in this respect PBVI is an approximation technique. As more belief points are added, the quality of optimization increases at the expense of additional computational complexity, allowing trade-offs to be made between optimization quality and computational complexity [85, 111].⁹

PBVI first samples a set of N belief points b_n using a *random policy*, described in Algorithm 2. In effect, actions are taken at random and the belief points encountered are noted. In theory it is possible that any belief point might eventually be reached starting from b_0 , but in practice it seems this is rarely the case and the belief point selection process used here attempts to find those belief points which are likely to be reached.¹⁰ Then, value iteration is performed, but optimized only for the sampled points, as shown in Algorithm 3. Like exact value iteration PBVI produces a set of vectors $\{V^n\}$ and corresponding actions $\{a^n\}$, but unlike exact value

⁸Technically it is the *complexity* of optimal policies, and not the number of states, actions, and observations which causes value iteration to become intractable, but it is not obvious how to calculate the complexity of a plan *a priori* and in practice the number of states, actions, and observations is a useful heuristic.

⁹The phrase “Point-based value iteration” and acronym PBVI were coined by Pineau to describe an algorithm which performs point-based back-ups on a set of points which is *grown* at each solution iteration [85]. Subsequent work such as the PERSEUS algorithm [111] has shown that using a *fixed* set of points also produces good policies, and since using a fixed set is simpler to describe, in this work PBVI is used to refer to point-based back-ups on a *fixed* set of points.

¹⁰The version of belief point sampling shown in Algorithm 2 assumes the POMDP is episodic and will, at the end of each episode, *reset* to its initial belief state. The VOICEMAIL POMDP has this property in that its *doSave* and *doDelete* actions reset to the initial belief state. If a POMDP lacks this property, then periodic resets will need to be added to Algorithm 2.

iteration the number of vectors produced in each iteration is constant, because each vector V^n corresponds to a belief point b_n in $\hat{\mathcal{B}}$. Although the conditional plan found for belief point b_n is only guaranteed to be optimal for that belief point, the hope is that it will be optimal, or nearly so, at other points nearby. At runtime, an optimal action a may be chosen for any belief point b by evaluating $a = a_n$ where $n = \arg \max_n \sum_s b(s)V^n(s)$, just as in exact value iteration.

Algorithm 2: Belief point selection for PBVI, using a random policy. For definitions of “randInt” and “sampleDist” see page xii.

Input: \mathfrak{B} , ϵ , N
Output: $\{b_n\}$

```

1  $s \leftarrow \text{sampleDist}_s(b_0)$ 
2  $n \leftarrow 1$ 
3  $b \leftarrow b_0$ 
4  $b_1 \leftarrow b$ 
5 while  $n < N$  do
    // Take a random action and compute new belief state.
6    $a \leftarrow \text{randInt}(|\mathcal{A}|)$ 
7    $s' \leftarrow \text{sampleDist}_{s'}(P(s'|s, a))$ 
8    $o' \leftarrow \text{sampleDist}_{o'}(P(o'|s', a))$ 
9    $b \leftarrow \text{SE}(b, a, o')$ 
    // If this is a (sufficiently) new point, add it to  $\hat{\mathcal{B}}$ .
    // Parameter  $\epsilon$  ensures points are spread out in belief space.
10  if  $\min_{i \in [1, n]} |b_i - b| > \epsilon$  then
11     $n \leftarrow (n + 1)$ 
12     $b_n \leftarrow b$ 
13   $s \leftarrow s'$ 

```

As compared to exact value iteration, PBVI optimization is faster for two reasons. First, because PBVI can enumerate $b_n^{a, o'}$ (the belief point reached when starting in point b_n , taking action a , and receiving observation o'), only optimal values of V_{t-1} are considered: unlike exact value iteration which generates $|\mathcal{A}||\mathcal{V}_{t-1}|^{|\mathcal{O}|}$ possibly useful conditional plans in each iteration, PBVI generates only $N \cdot |\mathcal{A}||\mathcal{O}|$ possibly useful conditional plans in each iteration. Second, in the pruning step, whereas exact value iteration must consider all $b \in \mathcal{B}$, PBVI only considers N points. Thus in PBVI, pruning is implemented as a simple $\arg \max$ operation. However, because PBVI only optimizes policies for a set of belief points (and not the entire belief simplex), it is possible that PBVI will fail to construct conditional plans for belief points not in \mathcal{B} . Policy quality may be assessed empirically by running successive optimizations with increasing numbers of belief points and determining whether an asymptote appears.¹¹

¹¹Analytical bounds may be stated if the belief point sampling procedure is changed; see [85].

Algorithm 3: Point-based value iteration (PBVI)

```

Input:  $\mathfrak{B}, \{b_n\}, T$ 
Output:  $\{V_T^n\}, \{a_T^n\}$ 
1 for  $n \leftarrow 1$  to  $N$  do
2   foreach  $s \in \mathcal{S}$  do
3      $V_0^n(s) \leftarrow 0$ 
4 for  $t \leftarrow 1$  to  $T$  do
5   // Generate  $\{v^{a,n}\}$ , the set of possibly useful conditional plans.
6   for  $n \leftarrow 1$  to  $N$  do
7     foreach  $a \in \mathcal{A}$  do
8       foreach  $o' \in \mathcal{O}$  do
9          $b_n^{a,o'} \leftarrow SE(b_n, a, o')$ 
10         $l(o') \leftarrow \arg \max_{\tilde{n}} \sum_{s'} b_n^{a,o'}(s') V_{t-1}^{\tilde{n}}(s')$ 
11        foreach  $s \in \mathcal{S}$  do
12           $v^{a,n}(s) \leftarrow r(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_{o'} P(o'|s', a) V_{t-1}^{l(o')}(s')$ 
13        // Prune  $\{v^{a,n}\}$  to yield  $\{V_t^n\}$ , set of actually useful conditional plans.
14        for  $n \leftarrow 1$  to  $N$  do
15           $a_t^n \leftarrow \arg \max_a \sum_s b_n(s) v^{a,n}(s)$ 
16          foreach  $s \in \mathcal{S}$  do
17             $V_t^n(s) \leftarrow v^{a_t^n, n}(s)$ 

```

Example of PBVI value iteration in the VOICEMAIL-2 application

To demonstrate PBVI, the VOICEMAIL application will be extended slightly. This extended POMDP will be called VOICEMAIL-2 and its observations will include a notion of ASR confidence score, which indicates, on a scale of 1 to 4, the likelihood a recognition hypothesis is correct. The observation set for the VOICEMAIL-2 POMDP is composed of eight elements ($|\mathcal{O}| = 8$) and $\mathcal{O} = \{\overline{save_1}, \overline{save_2}, \overline{save_3}, \overline{save_4}, \overline{delete_1}, \overline{delete_2}, \overline{delete_3}, \overline{delete_4}\}$. The observation function, given in Table 2.4, encodes the intuition that speech recognition is more likely to be accurate when the confidence score is high. In all other respects (including the state set, action set, transition function, and reward function), the VOICEMAIL-2 POMDP is identical to the VOICEMAIL POMDP.

Ten belief points ($N = 10$) were sampled using a random policy, as in Algorithm 2, to form $\hat{\mathcal{B}}$. Then, optimization was performed as in Algorithm 3 using a horizon of $T = 100$. The resulting value function and policy are shown in Figure 2.11. As in Figure 2.9, the (heavy) dashed lines show the three resulting partitions of the optimal policy. The dots on the b axis show the belief points used for optimization (i.e., the elements of $\hat{\mathcal{B}}$), and the (light) dotted lines rising up from these dots indicate their corresponding vector. Note that the same conditional plan starting with the *doSave* action is optimal for the three left-most belief points.

a	s'	$o' = \overline{save}_c$				$o' = \overline{delete}_c$			
		$c = 1$	$c = 2$	$c = 3$	$c = 4$	$c = 1$	$c = 2$	$c = 3$	$c = 4$
ask	$save$	0.08	0.16	0.24	0.32	0.08	0.06	0.04	0.02
	$delete$	0.12	0.09	0.06	0.03	0.07	0.14	0.21	0.28
$doSave$	$save$	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
	$delete$	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
$doDelete$	$save$	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
	$delete$	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125

Table 2.4 Observation function $P(o'|a, s')$ for the example VOICEMAIL-2 spoken dialog system POMDP. As in the VOICEMAIL application, the observation o' only conveys useful information following an ask action. Note that a higher value of c indicates ASR is more reliable.

To demonstrate the improvement in efficiency of PBVI over exact value iteration, both optimization techniques were run on this problem on the same computer. A standard implementation of PBVI (including sampling 10 points and 100 iterations of optimization) ran in approximately 1 second whereas exact value iteration ran for over 4 hours on the same computer without reaching 100 optimizations.¹²

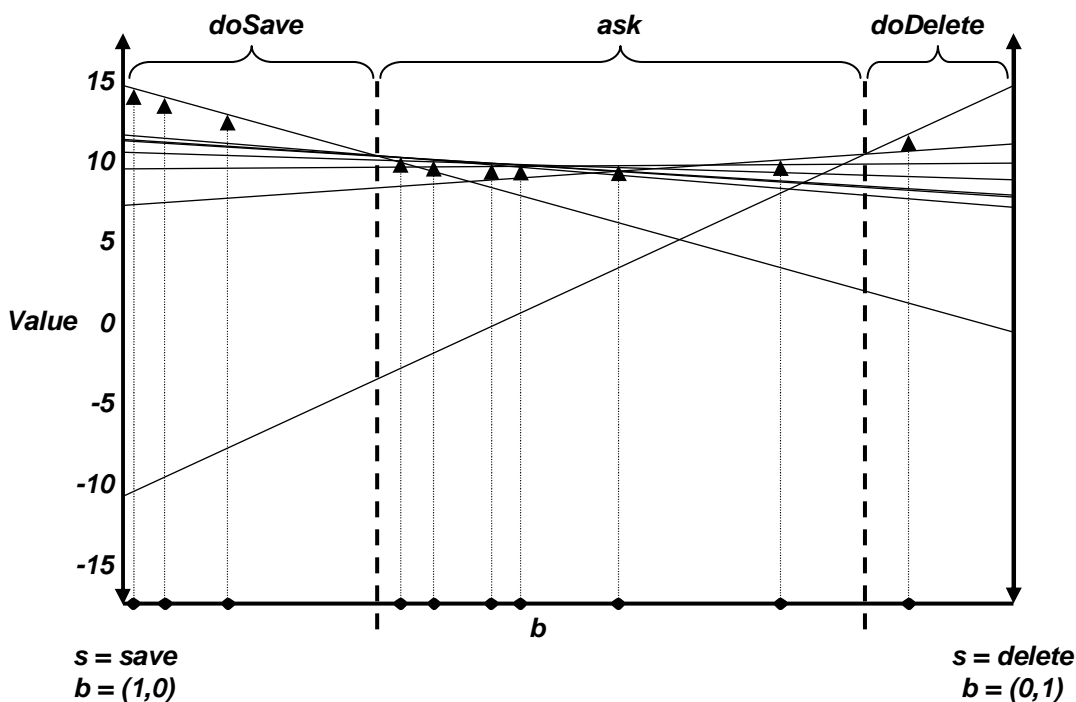


Figure 2.11 Example policy created with PBVI for the VOICEMAIL-2 POMDP. The dots on the b axis show the location of the belief points in \hat{B} used for optimization, and the (light) dotted arrows indicate which vector corresponds to each point. The (heavy) dashed lines show partitions where all conditional plans begin with the same action.

In summary, conditional plans provide a framework for evaluating different courses of ac-

¹²For exact value iteration, a standard package was used with its default settings [16].

tion, but enumerating all possible conditional plans is hopelessly intractable. Value iteration builds conditional plans incrementally for longer and longer time horizons, discarding useless plans as it progresses, making policy optimization possible for small POMDPs like the VOICEMAIL application. Even so, optimization for the slightly larger VOICEMAIL-2 application is intractable with value iteration. Point-based value iteration (PBVI), which finds optimal conditional plans for a fixed set of belief points, easily finds a good (but approximate) policy for this problem.

Although the example problems used in this chapter were inspired by spoken dialog systems, they side-stepped many challenges faced by real-world applications. The next chapter examines spoken dialog systems in detail, and shows how the POMDP framework can be extended to create a principled model of control in human-computer dialog.

Dialog management as a POMDP

In this chapter, a statistical model of spoken dialog called the “SDS-POMDP” is presented. This model is based on a partially observable Markov decision process in which the true state of the dialog is viewed as unobserved variable, and the speech recognition result is viewed as an observation. Rather than maintaining one hypothesis for the state of the dialog, the dialog model maintains a distribution over *all possible* dialog states. Based on this distribution and a reward function supplied by a dialog designer, the machine chooses actions to maximize the sum of rewards over the course of a dialog.

The chapter first formalizes the core components of a spoken dialog system, then details the SDS-POMDP model. After, an example spoken dialog system called the TRAVEL application is presented and its optimization and operation are discussed.

3.1 Components of a spoken dialog system

The architecture of a spoken dialog system is shown in Figure 3.1 [128]. In this depiction, the user has some internal state $s_u \in \mathcal{S}_u$ which corresponds to a goal that a user is trying to accomplish, such as a flight or bus itinerary, criteria for a conference room booking, or a desired computer purchase [120, 90, 12, 83]. s_u represents the user’s goal at a particular time and may change over the course of a dialog; \mathcal{S}_u is the set of all s_u . Also, from the user’s viewpoint, the dialog history has state $s_d \in \mathcal{S}_d$ which indicates, for example, what the user has said so far, or the user’s view of what has been *grounded* in the conversation so far [19, 113]. Based on the user’s goal at the beginning of each turn, the user takes some communicative action (also called an intention) $a_u \in \mathcal{A}_u$. In the literature, communicative actions are often separated into two components: *illocutionary force* and *propositional content* [4, 100]. *Illocutionary force* refers to the type of action such as *suggesting*, *requesting*, or *informing*, and are sometimes referred to as *speech acts* [4, 100] or *dialog acts* [21, 48]. *Propositional content* refers to the information contained in the action such as “Boston” or “10:00 AM” and can be represented by name-value pairs or hierarchical structures [18, 22, 38]. Each element a_u is a complete description of one possible user action and includes *both* of these elements, for example “suggest(10:00 AM)” or

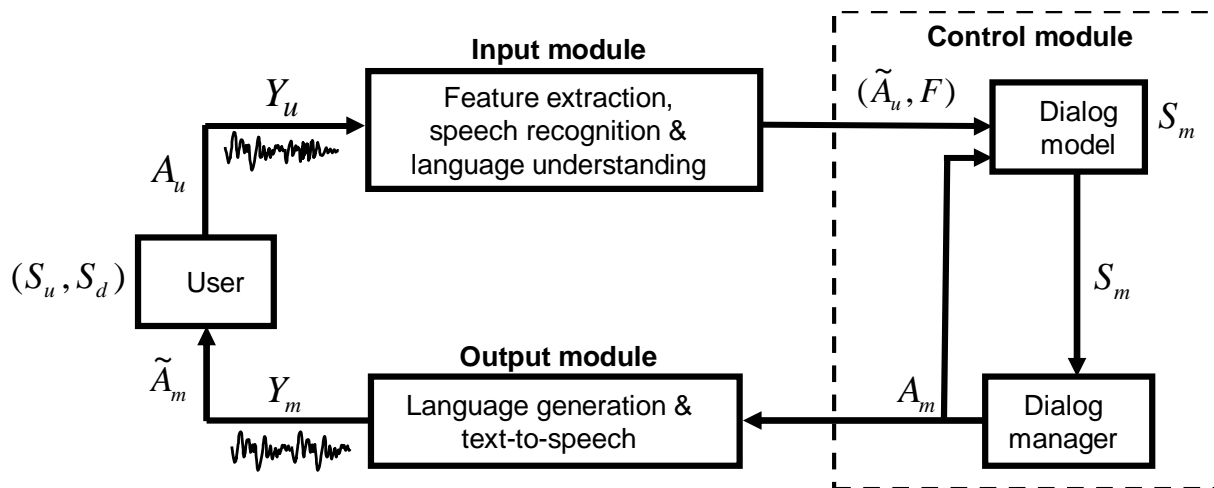


Figure 3.1 Typical architecture of a spoken dialog system. The control module, shown by the dotted box, contains the dialog model and dialog manager.

“request(destination)”.

The user renders a_u by choosing words to express this intention, then speaking them which produces an audio signal y_u . The input module, which performs speech recognition and language understanding, then takes the audio signal y_u and produces two outputs: \tilde{a}_u and f . The *recognition hypothesis* \tilde{a}_u is a (possibly incorrect) estimate of the user’s action a_u and is drawn from the same set as a_u , $\tilde{a}_u \in \mathcal{A}_u$. f provides one or more features of the recognition process and is drawn from the set $f \in \mathcal{F}$. These features might include discrete quantities such as number of words recognized, or real-valued quantities such as utterance duration. The details of the specific recognition and understanding process used are not important to this model (since any technique will introduce errors), but interested readers are referred to texts such as [50] or survey papers such as [134, 33] for details and references.

The recognition hypothesis \tilde{a}_u and recognition features f are then passed to the control module. Inside the control module, the dialog model, which maintains an internal state $s_m \in \mathcal{S}_m$ that tracks the state of the conversation from the perspective of the machine, updates its s_m using \tilde{a}_u and f . s_m is then passed to the dialog manager, which decides what action $a_m \in \mathcal{A}_m$ the machine should take based on s_m . a_m is converted to an audio response y_m by the output module using language generation and text-to-speech. a_m is also passed back to the dialog model so that s_m may track both user and machine actions. The user listens to y_m , attempts to recover a_m , and as a result might update their goal state s_u and their interpretation of the dialog history s_d . For example, if the user’s goal is to take a direct train from London to Berlin, and the system says “There aren’t any trains from London to Berlin”, the user’s goal might change to include indirect trains, or to include flights. After the user’s state is updated, the cycle then repeats.

One key reason why spoken dialog systems are challenging to build is that \tilde{a}_u will contain recognition errors: i.e., it is frequently the case that $\tilde{a}_u \neq a_u$. Indeed, sentence error rates in

the DARPA Communicator project ranged from 11.2% to 42.1% across 9 sites [120]. Worse, as systems which attempt to handle more complex dialog phenomena are created, users' speech also becomes more complex, exacerbating recognition errors. As a result, the user's action a_u , the user's state s_u , and the dialog history s_d are not directly observable and can never be known to the system with certainty. However, intuitively \tilde{a}_u and f provide evidence from which a_u , s_u , and s_d can be *inferred*, and the next section formalizes this as a POMDP.

3.2 The SDS-POMDP model

A spoken dialog system will now be cast as a POMDP. First, the machine action a_m will be cast as the POMDP action a . In a POMDP, the POMDP state s expresses the unobserved state of the world and the above analysis suggests that this unobserved state can naturally be factored into three distinct components: the user's goal s_u , the user's action a_u , and the dialog history s_d . Hence, the factored POMDP state \mathcal{S} is defined as:

$$s = (s_u, a_u, s_d) \quad (3.1)$$

and the system state s_m becomes the belief state b over s_u , a_u , and s_d :

$$s_m = b(s) = b(s_u, a_u, s_d). \quad (3.2)$$

This factored form will henceforth be referred to as the SDS-POMDP (spoken dialog system partially observable Markov decision process).

The noisy recognition hypothesis \tilde{a}_u and the recognition features f will then be cast as the SDS-POMDP observation o :

$$o = (\tilde{a}_u, f). \quad (3.3)$$

To compute the transition function and observation function, a few intuitive assumptions will be made. First, Equation 3.1 is substituted into the POMDP transition function and decomposed:

$$\begin{aligned} P(s'|s, a) &= P(s'_u, s'_d, a'_u | s_u, s_d, a_u, a_m) \\ &= P(s'_u | s_u, s_d, a_u, a_m) P(a'_u | s'_u, s_u, s_d, a_u, a_m) P(s'_d | a'_u, s'_u, s_u, s_d, a_u, a_m). \end{aligned} \quad (3.4)$$

Conditional independence will then be assumed as follows. The first term in Equation 3.4, called the *user goal model* \mathcal{T}_{S_u} , indicates how the user's goal changes (or does not change) at each time-step. It is assumed that the user's goal at each time-step depends only on the previous goal, the dialog history, and the machine's action:

$$\mathcal{T}_{S_u} = P(s'_u | s_u, s_d, a_u, a_m) = P(s'_u | s_u, s_d, a_m). \quad (3.5)$$

The second term, called the *user action model* \mathcal{T}_{A_u} , indicates what actions the user is likely to take at each time step. It is assumed the user's action depends on their (current) goal, the dialog history, and the preceding machine action:

$$\mathcal{T}_{A_u} = P(a'_u | s'_u, s_u, s_d, a_u, a_m) = P(a'_u | s'_u, s_d, a_m). \quad (3.6)$$

The third term, called the *dialog history model* \mathcal{T}_{S_d} , captures relevant historical information about the dialog. This component has access to the most recent value of all variables:

$$\mathcal{T}_{S_d} = P(s'_d | a'_u, s'_u, s_u, s_d, a_u, a_m) = P(s'_d | a'_u, s'_u, s_d, a_m). \quad (3.7)$$

Substituting Equations 3.5, 3.6, and 3.7 into 3.4 then gives the SDS-POMDP transition function:

$$P(s' | s, a) = P(s'_u | s_u, s_d, a_m) P(a'_u | s'_u, s_d, a_m) P(s'_d | a'_u, s'_u, s_d, a_m). \quad (3.8)$$

From Equations 3.1 and 3.3, the observation function of the SDS-POMDP becomes:

$$P(o' | s', a) = p(\tilde{a}'_u, f' | s'_u, s'_d, a'_u, a_m). \quad (3.9)$$

The observation function accounts for the corruption introduced by the speech recognition and language understanding process, so it is assumed that the observation depends only on the action taken by the user:¹

$$P(o' | s', a) = p(\tilde{a}'_u, f' | a'_u). \quad (3.10)$$

The two equations 3.8 and 3.10 represent a statistical model of a spoken dialog system. The transition function allows future behaviour to be predicted and the observation function provides the means for inferring a distribution over hidden user states from observations. The models themselves have to be estimated of course. The user goal model and the user action model (the first two components of Equation 3.8), can be estimated from a corpus of annotated interactions: for example, a corpus could be annotated, and conditional distributions over user dialog acts can be estimated given a machine dialog act and a user goal. To appropriately cover all of the conditions, the corpus used would need to include variability in the strategy employed by the machine – for example, using a Wizard-of-Oz framework with a simulated ASR channel [112, 106], as will be done later in this thesis (Chapters 5 and 6).

The dialog history model (the final term of Equation 3.8) can either be estimated from data, or handcrafted as either a finite state automaton (as in for example [113]) or more complex rules such as the “Information State Update” approach [59]. Thus the SDS-POMDP system dynamics enable both probabilities estimated from corpora and hand-crafted heuristics to be incorporated. This is a very important aspect of the SDS-POMDP framework in that it allows deterministic programming to be incorporated alongside stochastic models in a straightforward way.

The observation function can be estimated from a corpus or derived analytically using a model of the speech recognition process [25, 112]. The set of features F can contain both discrete and continuous variables. In practice, and especially if continuous features are used, it is likely that approximations to this model will be required, examples of which are shown later in this chapter, and in Chapters 5 and 6.

¹Here it is assumed that the user can say anything at any point in the dialog and one language model is used throughout. If the machine has the ability to change recognition grammars (via a machine action), then a dependence on a_m can be added to this conditional probability table.

Finally, given the definitions above, the belief state can be updated at each time step by substituting equations 3.8 and 3.10 into 2.3 and simplifying:

$$b'(s'_u, s'_d, a'_u) = \eta \cdot p(\tilde{a}'_u, f | a'_u) \sum_{s_d} P(a'_u | s'_u, s_d, a_m) \sum_{s_u} P(s'_u | s_u, s_d, a_m) \cdot P(s'_d | a'_u, s'_u, s_d, a_m) \sum_{a_u} b(s_u, s_d, a_u). \quad (3.11)$$

The summations over $s = (s_u, a_u, s_d)$ predict a new distribution for s' based on the previous values weighted by the previous belief. For each assumed value of a'_u , the leading terms outside the summation scale the updated belief by the probability of the observation given a'_u and the probability that the user would utter a'_u given the user's goal and the last machine output.

The reward function is not specified explicitly since it depends on the design objectives of the target system. The reward function may contain incentives for dialog speed by using a per-turn penalty and dialog "appropriateness", for example by including a penalty for confirming an item which has not been discussed yet. Also note that conditioning rewards on task completion is straightforward in the SDS-POMDP since the state space explicitly contains the user's goal. With the addition of a reward function the POMDP is fully specified and can be optimized to produce a policy $\pi(b)$ which serves as the dialog manager, choosing actions based on the belief state over s_u, s_d , and a_u .

For ease of reference, Table 3.1 summarises the terms in a standard POMDP and their expansions in the SDS-POMDP model, and Figure 3.2 shows the SDS-POMDP model as an influence diagram [49].² The tuple defining an SDS-POMDP will be written \mathfrak{P}_{SDS} where $\mathfrak{P}_{SDS} = (\mathcal{S}_u, \mathcal{A}_u, \mathcal{S}_d, \mathcal{A}_m, \mathcal{T}_{s_u}, \mathcal{T}_{a_u}, \mathcal{T}_{s_d}, \mathcal{R}, \mathcal{Z}, \gamma, b_0)$.

	Standard POMDP	SDS-POMDP
State	s	(s_u, a_u, s_d)
Observation	o	(\tilde{a}_u, f)
Action	a	a_m
Transition function	$P(s' s, a)$	$P(s'_u s_u, s_d, a_m) P(a'_u s'_u, s_d, a_m) P(s'_d a'_u, s'_u, s_d, a_m)$
Observation function	$P(o' s', a)$	$p(\tilde{a}'_u, f' a'_u)$
Reward function	$r(s, a)$	$r(s_u, a_u, s_d, a_m)$
Belief state	$b(s)$	$b(s_u, a_u, s_d)$

Table 3.1 Summary of SDS-POMDP components.

In sum, the SDS-POMDP model allows the dialog management problem to be cast in a statistical framework, and it is therefore particularly well-suited to coping with the uncertainty inherent in human/machine spoken dialog. The user's goal, action, and relevant dialog history are cast as unobserved variables, and as a dialog progresses the machine maintains an evolving distribution over all possible combinations of these variables. The update of this distribution combines evidence from the speech recognizer with background knowledge in the form of models of the

²In the next chapter, alternate approaches to dialog management are also presented as influence diagrams.

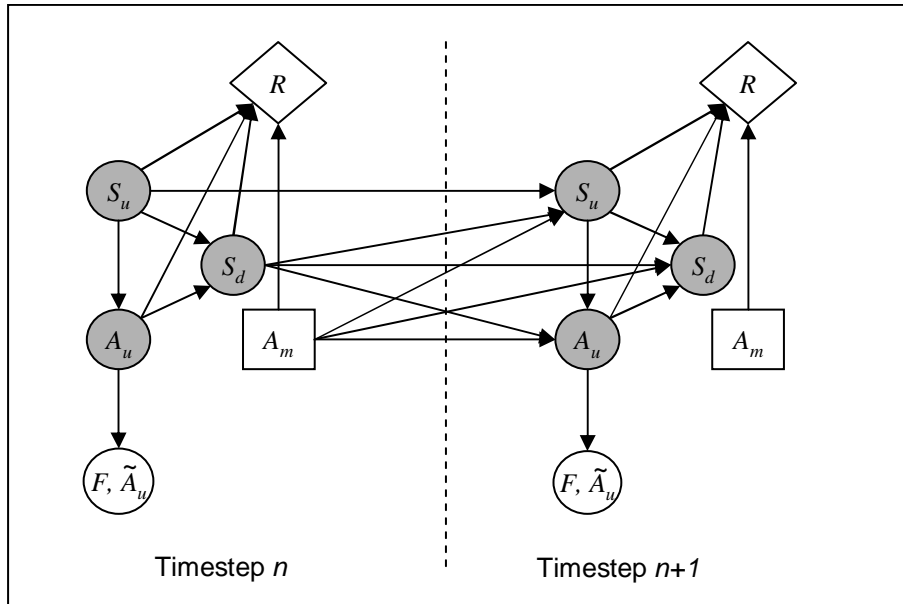


Figure 3.2 SDS-POMDP model shown as an influence diagram [49]. Shaded circles represent unobserved (hidden) random variables; un-shaded circles represent observed random variables; squares represent decision nodes; and diamonds represent reward (utility) nodes. Arrows show causal influence.

user’s behavior, the speech recognition process, and deterministic heuristics. A (human) dialog designer provides objectives in the form of a reward function, and actions are chosen to maximize the sum of rewards over time using POMDP optimization.

The remainder of this chapter illustrates the SDS-POMDP model in some detail with an example spoken dialog system called the TRAVEL application.

3.3 Example SDS-POMDP application: TRAVEL

In this section an example TRAVEL application in the SDS-POMDP framework is presented. This application will be used to illustrate the SDS-POMDP framework in this chapter, and to make quantitative comparisons between the SDS-POMDP approach and existing approaches in the next chapter.

In the TRAVEL application, a user is trying to buy a ticket to travel from one city to another city, with the set of cities given as \mathcal{C} . The user’s goal is given as

$$s_u = (x, y) : x \in \mathcal{C}, y \in \mathcal{C}, x \neq y. \quad (3.12)$$

In words, the user’s goal consists of an itinerary from x to y where x and y are both cities in the set \mathcal{C} and x and y are different cities, such as London and Edinburgh. The total number of user goals $|\mathcal{S}_u|$ is $|\mathcal{C}| \cdot |\mathcal{C} - 1|$, and the initial distribution over user goals is uniform; i.e.,

$$b_0(s_u) = \frac{1}{|\mathcal{S}_u|} : s_u \in \mathcal{S}_u. \quad (3.13)$$

It is assumed that the user’s goal is fixed throughout the dialog, and user goal model is thus defined accordingly:

$$P(s'_u | s_u, s_d, a_m) = P(s'_u | s_u) = \begin{cases} 1, & \text{if } s'_u = s_u, \\ 0, & \text{otherwise} \end{cases} \quad (3.14)$$

The machine asks a series of questions about where the user wants to travel from and to, and then “submits” a ticket purchase request, which prints the ticket and ends the dialog. The machine may also choose to “fail,” abandoning the dialog. The machine action set \mathcal{A}_m is shown in Table 3.2.

a_m	Meaning
<i>greet</i>	Greets the user and asks “How can I help you?”
<i>ask-from</i>	Asks the user where they want to leave from.
<i>ask-to</i>	Asks the user where they want to go to.
<i>conf-from-x</i>	Confirm that the user wants to leave from city $x \in \mathcal{C}$.
<i>conf-to-y</i>	Confirm that the user wants to go to city $y \in \mathcal{C}$.
<i>submit-x-y</i>	Closes the dialog and prints a ticket from city x to city y .
<i>fail</i>	Abandons the dialog without printing a ticket.

Table 3.2 *Machine actions in the TRAVEL application.*

The user action set \mathcal{A}_u include various ways of partially or fully communicating the user’s itinerary, and saying “yes” and “no”. The set of user actions is given in Table 3.3.

a_u	Meaning
<i>from-x</i>	Intention “I want to leave from x ”, $x \in \mathcal{C}$.
<i>to-x</i>	Intention “I want to go to x ” $x \in \mathcal{C}$.
x	Intention “ x ”, $x \in \mathcal{C}$, with no indication of to/from.
<i>from-x-to-y</i>	Intention “I want to go from x to y ” $x \in \mathcal{C}, y \in \mathcal{C}$
<i>yes</i>	Intention “yes” (response to a yes/no question)
<i>no</i>	Intention “no” (response to a yes/no question)
<i>null</i>	The user did not respond (i.e., the user said nothing)

Table 3.3 *User actions in the TRAVEL application.*

The user action model $P(a'_u | s_d, s'_u, a_m)$ is defined so that the user responds with honest but varied responses. For example, the user responds to $a_m = \textit{ask-to}$ with (for example) $a'_u = \textit{edinburgh}$, $a'_u = \textit{to-edinburgh}$, $a'_u = \textit{from-london-to-edinburgh}$, or $a'_u = \textit{null}$. For simplicity the dependency on s_d has been ignored – i.e., it has been assumed that the user’s response depends on their goal and the machine’s most recent action only. The probabilities themselves were handcrafted, selected based on experience performing usability testing with slot-filling dialog systems. The user action model is summarized in Table 3.4.

As above, the observation is formed of two elements, \tilde{a}'_u and f , and the recognition hypothesis \tilde{a}'_u is drawn from the set of user actions $\tilde{a}'_u \in \mathcal{A}_u$. Recognition results are represented at

Machine Action		User Response		$P(a_u' s_u' = \text{from-london-to-edinburgh}, a_m)$
Utterance	a_m	Utterance	a_u'	
“Hi, how can I help?”	<i>greet</i>	“From London to Edinburgh”	<i>from-london-to-edinburgh</i>	0.540
		“Leaving from London”	<i>from-london</i>	0.180
		“Going to Edinburgh”	<i>to-edinburgh</i>	0.180
		(user says nothing)	<i>null</i>	0.100
“Where are you leaving from?”	<i>ask-from</i>	“London”	<i>london</i>	0.585
		“From London”	<i>from-london</i>	0.225
		“From London to Edinburgh”	<i>from-london-to-edinburgh</i>	0.090
		(user says nothing)	<i>null</i>	0.100
“Where are you going to?”	<i>ask-to</i>	“Edinburgh”	<i>edinburgh</i>	0.585
		“To Edinburgh”	<i>to-edinburgh</i>	0.225
		“From London to Edinburgh”	<i>from-london-to-edinburgh</i>	0.090
		(user says nothing)	<i>null</i>	0.100
“To Edinburgh, is that right?”	<i>confirm-to-edinburgh</i>	“Yes”	<i>yes</i>	0.765
		“Edinburgh”	<i>edinburgh</i>	0.101
		“To Edinburgh”	<i>to-edinburgh</i>	0.034
		(user says nothing)	<i>null</i>	0.100
“To Cambridge, is that right?”	<i>confirm-to-edinburgh</i>	“No”	<i>no</i>	0.765
		“Edinburgh”	<i>edinburgh</i>	0.101
		“To Edinburgh”	<i>to-edinburgh</i>	0.034
		(user says nothing)	<i>null</i>	0.100

Table 3.4 Summary of user model parameters for the TRAVEL application. This example assumes that the user’s goal is (london,edinburgh). Due to space, user responses to $a_m = \text{confirm-from-x}$ are not shown; they follow analogous distributions to responses to $a_m = \text{confirm-to-x}$.

the concept level, and it is assumed that concept errors occur with probability p_{err} , and that all confusions are equally likely. Further, the simulated recognition process includes one recognition feature, *confidence score*, which provides an indication of the reliability of \tilde{a}'_u . The speech recognition model $P(\tilde{a}'_u, f' | a'_u)$ is in practice impossible to estimate directly from data, so it is decomposed into two distributions – one for “correct” recognitions and another for “incorrect” recognitions, yielding:

$$P(\tilde{a}'_u, c' | a'_u) = \begin{cases} p_h(c') \cdot (1 - p_{err}) & \text{if } \tilde{a}'_u = a'_u \\ p_h(1 - c') \cdot \frac{p_{err}}{|\mathcal{A}_u| - 1} & \text{if } \tilde{a}'_u \neq a'_u, \end{cases} \quad (3.15)$$

where c is defined on the interval $[0, 1]$. Past work has found that confidence score density broadly follows an exponential distribution [83], and here $p_h(c)$ is an exponential probability density function with slope determined by a parameter $h : h \geq 0$:

$$p_h(c) = \begin{cases} \frac{he^{hc}}{e^h - 1} & \text{if } h > 0 \\ 1 & \text{if } h = 0. \end{cases} \quad (3.16)$$

When $h = 0$, $p_h(c)$ is a uniform density and conveys no information; as h approaches (positive) infinity, $p_h(c)$ provides complete and perfect information.³ The effect of the parameter h is shown in Figure 3.3, which shows the probability density of confidence scores for correct and incorrect recognitions, scaled by the likelihood of mis-recognitions: i.e., the solid line shows $p(c, \text{correct recognition})$, and the dashed line shows $p(c, \text{incorrect recognition})$. In this figure, the concept error rate is set to $p_{err} = 0.3$. When $h = 0$ (Figure 3.3a), confidence scores are drawn from uniform distributions and convey no information. As h is increased from 1 to 5, the distributions of confidence scores for correct and incorrect recognitions become increasingly separated, such that correct and incorrect recognitions can be discerned with increasing accuracy. In the limit ($h = \infty$), the confidence score for correct recognitions is always 1, and for incorrect recognitions always 0, enabling correct and incorrect recognitions to be identified unambiguously. In modern spoken dialog systems, confidence score informativeness is broadly in the range of $h = 1$ to $h = 5$ [84], and this range will be used in the experiments throughout the thesis.

The dialog history s_d contains three components. Two of these indicate whether the from and to cities have been “not specified” (n), “unconfirmed” (u), or “confirmed” (c). “Not specified” means that the user has not referred to a city; “unconfirmed” means that the user has referred to a city once; and “confirmed” means that the user has referred to a city more than once. A third component, z , specifies whether the current turn is the first turn (1) or not (0). There are a total of 18 dialog states, given by:

$$s_d = (x, y, z) : x \in \{n, u, c\}, y \in \{n, u, c\}, z \in \{0, 1\}. \quad (3.17)$$

The dialog history model $P(s'_d | a'_u, s'_u, s_d, a_m)$ is defined to deterministically implement a notion of “grounding” from the user’s perspective. That is, a city which has not been referred to by the

³Note that $p_h(c)$ is a proper probability density in that, on the interval $[0, 1]$, $p_h(c) \geq 0$ and $\int_0^1 p_h(c) dc = 1$.

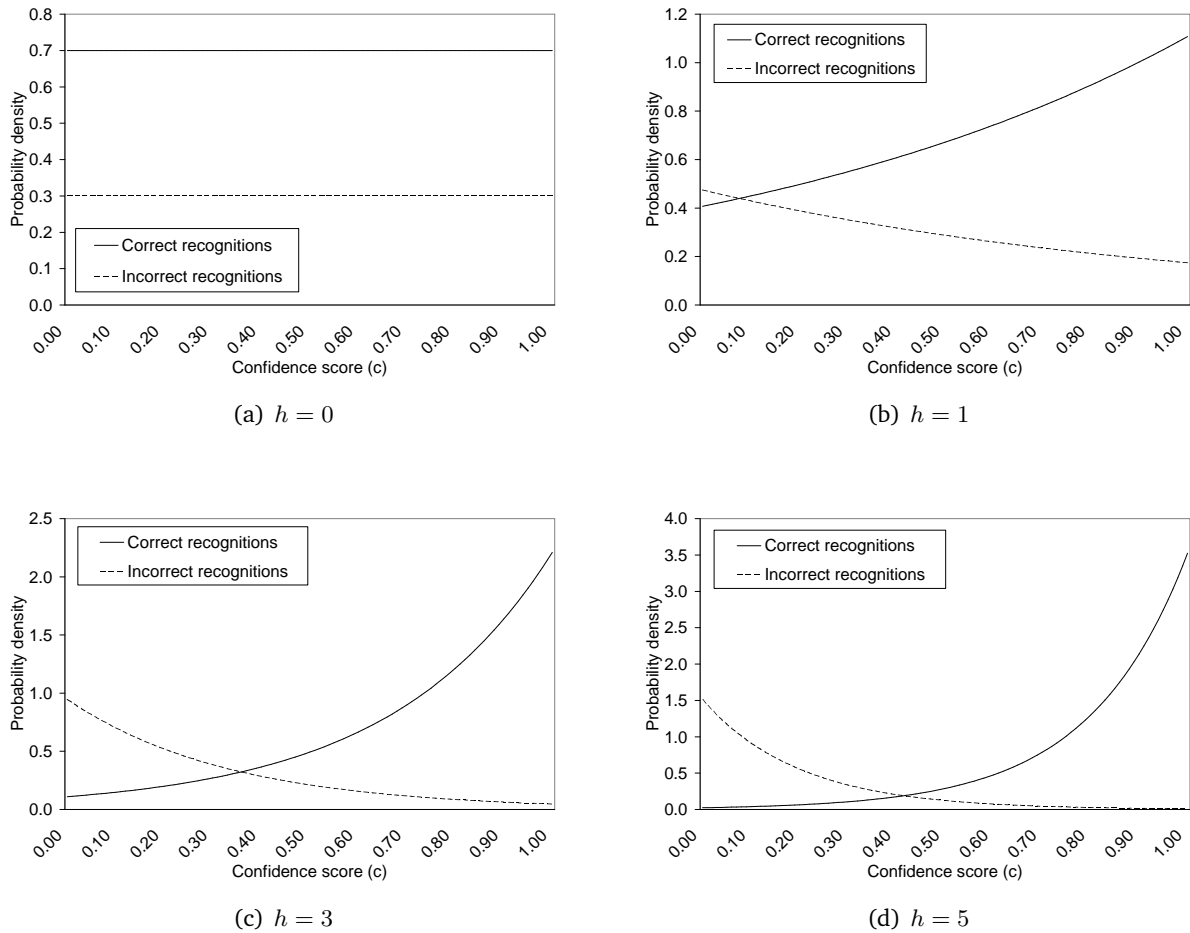


Figure 3.3 Probability density for various levels of confidence score informativeness (h), for concept error rate $p_{err} = 0.30$. In each plot, the solid line shows $p(c, \text{correct recognition})$ and the dotted line shows $p(c, \text{incorrect recognition})$. At $h = 0$ (upper left), the distribution of confidence scores is uniform and thus conveys no information about whether a mis-recognition has occurred. As h increases, the distributions become increasingly disjoint, which enables mis-recognitions to be identified more accurately.

user takes the value n ; a field which has been referred to by the user exactly once takes the value u ; and a field which has been referenced by the user more than once takes the value c .

Finally, to the set of states \mathcal{S} , a terminal absorbing state was added. When (and only when) the machine takes a *submit-x-y* or *fail* action, control transitions to this state.⁴

The reward measure includes components for both task completion and dialog “appropriateness”. For example, the reward for confirming an item which has been mentioned is -1 but for confirming an item which hasn’t been confirmed is -3 since the latter deviates from conversational norms. Finally, a reward for taking the *greet* action after the first turn is set to -100, which ensures that the machine will only consider taking the *greet* action in the first turn. Overall the reward measure reflects the intuition that behaving inappropriately or even abandoning a hopeless conversation early are both less severe than submitting the user’s goal incorrectly and

⁴This construction ensure that actions are chosen to maximize reward per-dialog, as opposed to per-turn over many dialogs.

printing the wrong ticket. The reward function is detailed in Table 3.5.

A discount of $\gamma = 0.95$ was selected, reflecting the intuition that actions should be selected with a relatively long horizon in mind.

3.4 Optimization of the TRAVEL application

With a POMDP defined, the next task is to find a policy. Two optimization methods were explored: (exact) value iteration and PBVI (Sections 2.3 and 2.4, pages 14 and 20). For exact value iteration, the publically available “solve-pomdp” package was used [16]; for PBVI the *Perseus* package was used [111].⁵ First, the smallest possible version of TRAVEL – 3 cities – was considered ($|C| = 3$, resulting in a total of 1945 states), without access to any useful confidence score information ($h = 0$).⁶ As expected, exact value iteration is unable to find a policy in a reasonable amount of time (1 iteration runs for more than 1 hour). For PBVI, policy quality scales with the number of belief points $|\hat{\mathcal{B}}|$ sampled, and with the number of iterations performed (Section 2.4). Experiments were run varying each of these parameters, described next.

Figure 3.4 shows the number of belief points N vs. the expected return for the initial belief state for various concept error rates (values of p_{err}) using 30 iterations of PBVI. In general, increasing the number of belief points increases average return up to an asymptote. For all of the speech recognition error rates considered, PBVI reached its maximum before 500 belief points, and henceforth all results reported for PBVI run on the 3-city TRAVEL SDS-POMDP use 500 belief points.

Solution quality in value iteration also increases as the planning horizon increases. Figure 3.5 shows number of iterations of PBVI vs. expected return for various concept error rates (values of p_{err}) using 500 belief points. Maximum return is reached well before 30 iterations, and henceforth all results reported for PBVI run on the 3-city TRAVEL SDS-POMDP use 30 iterations of PBVI.

Figure 3.6 shows concept error rate (p_{err}) vs. average return gained per dialog with the solid line and left axis, and vs. average dialog length with the dotted line on the right axis. As speech recognition errors become more prevalent, the average return decreases, and the average dialog length increases. This matches intuition: when speech recognition errors are more common, dialogs take longer and may also be less accurate.

Finally, Figure 3.7 shows an example conversation between a user and an SDS-POMDP policy in which the probability of making a concept error at each turn is $p_{err} = 0.3$. At each time-step, the machine action a_m and observation \tilde{a}'_u are used to update the belief state using Equation 3.11. The small graphs on the right-hand side show distributions over all values of user goals \mathcal{S}_u .

⁵Again here, PBVI refers to the general class of optimization techniques which use point-based back-ups and not specifically Pineau’s “Anytime” algorithm [85]. *Perseus* makes a few improvements to standard PBVI which are not important to this discussion and details can be found in [111].

⁶Using confidence score information is explored in section 4.2, starting on page 48.

User goal (s_u)	Machine action (a_m)	Dialog history (s_d)	Description of machine action in context	$r(s_u, s_d, a_m)$
—	<i>greet</i>	(—, —, 1) (—, —, 0)	Greet at first step of dialog Greet after first step of dialog	-1 -100
—	<i>ask-from</i>	(—, —, —)	Ask for <i>from</i> value	-1
—	<i>ask-to</i>	(—, —, —)	Ask for <i>to</i> value	-1
—	<i>confirm-from-london</i>	(u, —, —) ({n, c}, —, —)	Confirms <i>from</i> value which hasn't been stated yet Confirms <i>from</i> value which <i>has</i> been stated	-3 -1
—	<i>confirm-to-cambridge</i>	(—, u, —) (—, {n, c}, —)	Confirms <i>to</i> value which hasn't been stated yet Confirms <i>to</i> value which <i>has</i> been stated	-3 -1
(london, edinburgh)	<i>submit-london-edinburgh</i>	(—, —, —)	Submits correct value	+10
	<i>submit-cambridge-edinburgh</i>	(—, —, —)	Submits incorrect value (one slot)	-10
	<i>submit-cambridge-leeds</i>	(—, —, —)	Submits incorrect value (both slots)	-10
—	<i>fail</i>	(—, —, —)	Machine abandons conversation	-5

Table 3.5 Reward function for the TRAVEL application. The dash (—) placeholder indicates that the table row applies to any value.

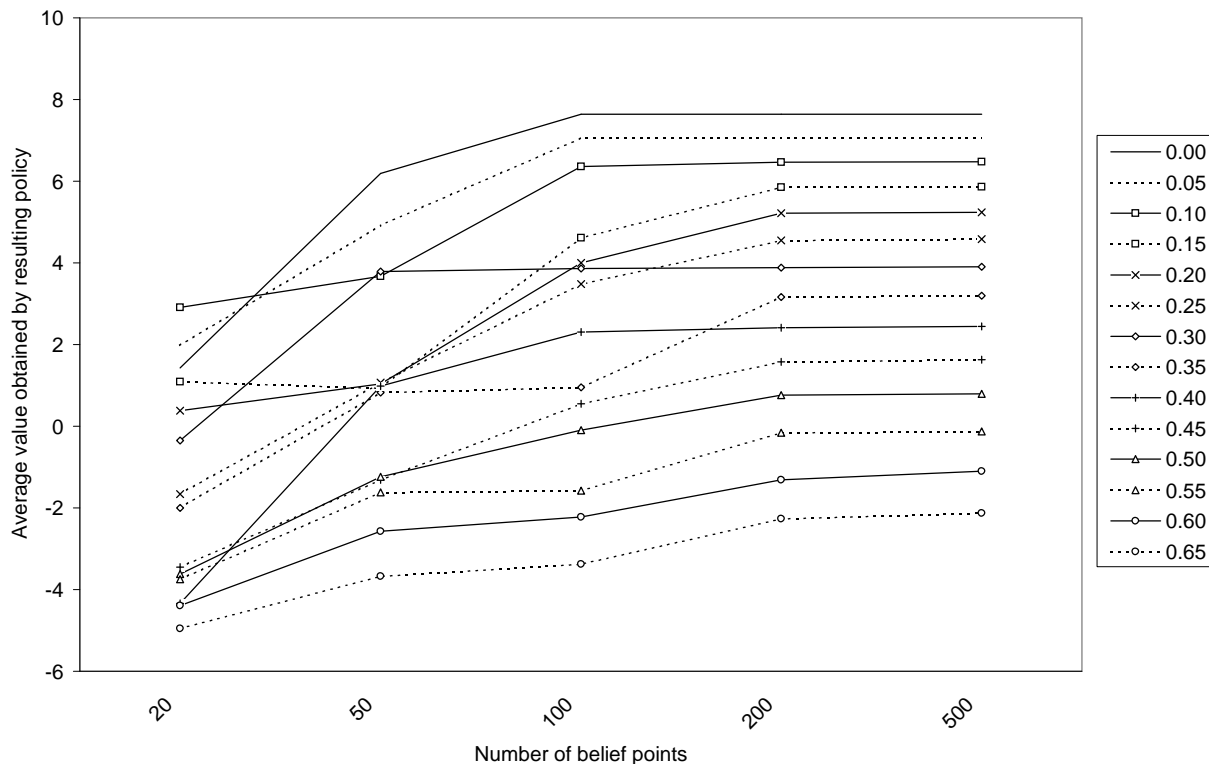


Figure 3.4 Number of belief points sampled by PBVI vs. average return for various concept error rates (values of p_{err}) for the 3-city TRAVEL SDS-POMDP. Increasing the number of belief points increases average return up to an asymptote, reached before 500 belief points for all concept error rates. Henceforth all experiments use 500 belief points.

At the beginning of the dialog, belief mass is distributed among all user goals equally, as specified by b_0 . At the opening of the dialog, the policy computed by optimization instructs the machine to take the *greet* action (“How can I help?”) in M1, and the recognition of “I’m leaving from Edinburgh” ($a_u = from-edinburgh$) in U1 is successful. Equation 3.11 is applied and belief mass shifts toward user goals which include “from Edinburgh”. However since “from Edinburgh” may have been the result of a speech recognition error, some mass remains over the other user goals.

This process is repeated at each time-step. In M2, the machine asks where the user is going to, and the reply of “to London” is *mis-recognized* as “Edinburgh”. Belief mass shifts towards goals which include “to Edinburgh”, and the exact distribution of belief mass again determined by Equation 3.11. At the end of U2, the goals (L \rightarrow C) and (C \rightarrow L) have very little belief mass, a consequence of neither of the preceding two observations supporting them.

In M3, the machine asks “Where are you leaving from?” and the response “Edinburgh” is correctly recognized. This observation shifts belief mass *back* towards user goals which include “from Edinburgh”. The user goals (L \rightarrow C) and (C \rightarrow L) now have negligible belief mass, since no observations have supported them. In M4 the machine asks “Where are you going to?” and the response “London” is recognized successfully. Of the user goals with non-negligible belief

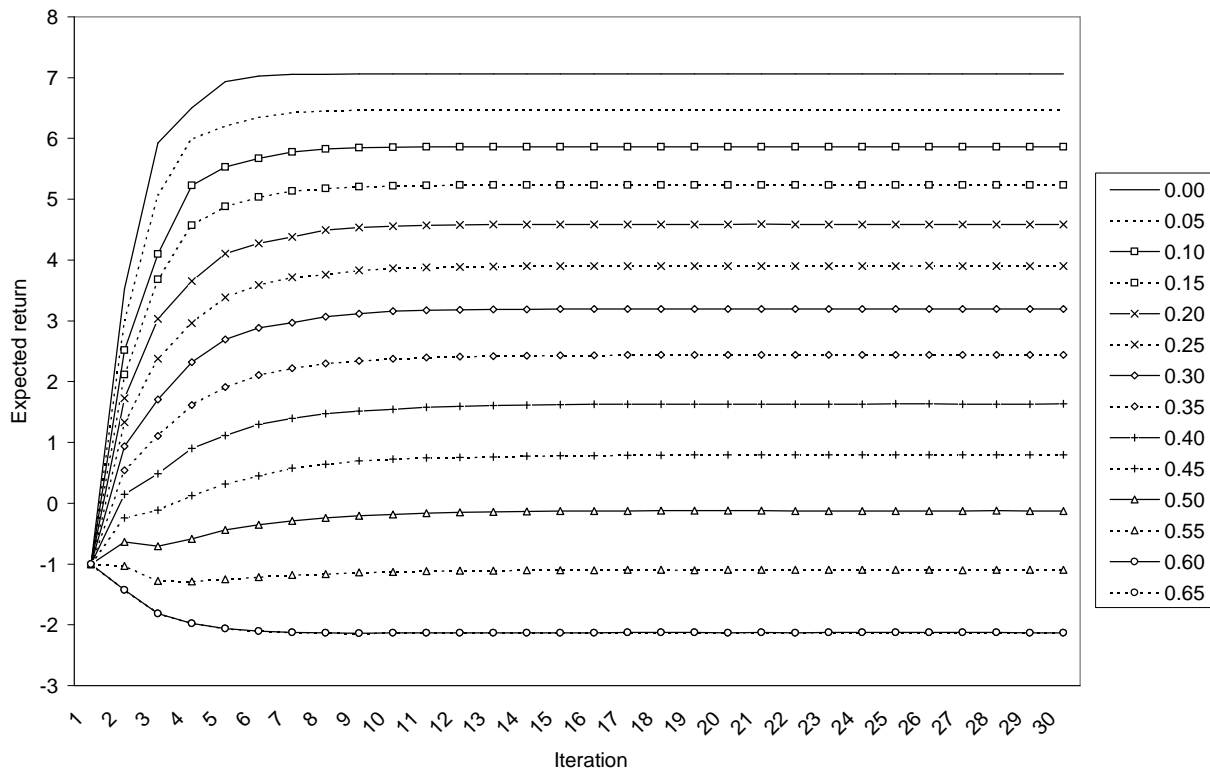


Figure 3.5 Number of iterations of PBVI vs. average return for various concept error rates (values of p_{err}) for the 3-city TRAVEL SDS-POMDP. Increasing the number of iterations increases average return up to an asymptote, reached before 30 iterations for all concept error rates. Henceforth all experiments use 30 iterations of PBVI.

mass remaining, only one is consistent with itineraries to London, and belief mass shifts toward the goal ($E \rightarrow L$). The resulting belief mass on ($E \rightarrow L$) is sufficient for the machine to print a ticket for this itinerary. In this example, the machine printed the correct ticket. Of course the machine does not always correctly identify the user’s goal, but by adjusting the reward measure, the dialog designer can specify the trade-off the system should make between the average length of the dialog and the accuracy desired.

Although the TRAVEL application with 3 cities is too small to be useful as a real dialog system, it nonetheless effectively illustrates the SDS-POMDP framework and captures the key challenges faced by a real dialog system. Further, good policies for this application can be produced with an existing, well-understood optimization technique: the *Perseus* implementation of PBVI. For these reasons, the 3-city version of the TRAVEL application is used in the next chapter (Chapter 4) to make comparisons between the SDS-POMDP model and existing techniques.

That said, the SDS-POMDP model as presented here faces important scalability challenges: a version of TRAVEL with four cities ($|C| = 4$, resulting in 5833 states) was also considered, and PBVI was not able to find acceptable policies in a reasonable amount of time for this version. Later in this thesis, Chapters 5 and 6 will address the key problem of how to scale the SDS-POMDP framework to problems of a realistic size.

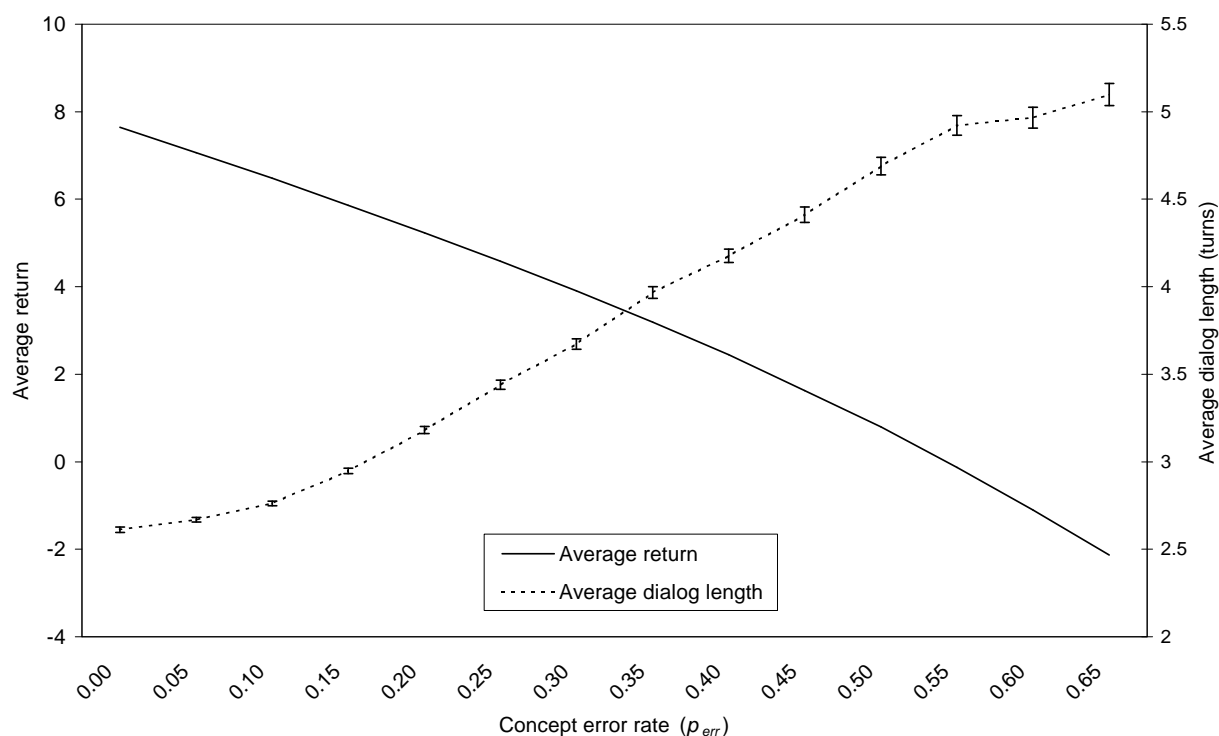
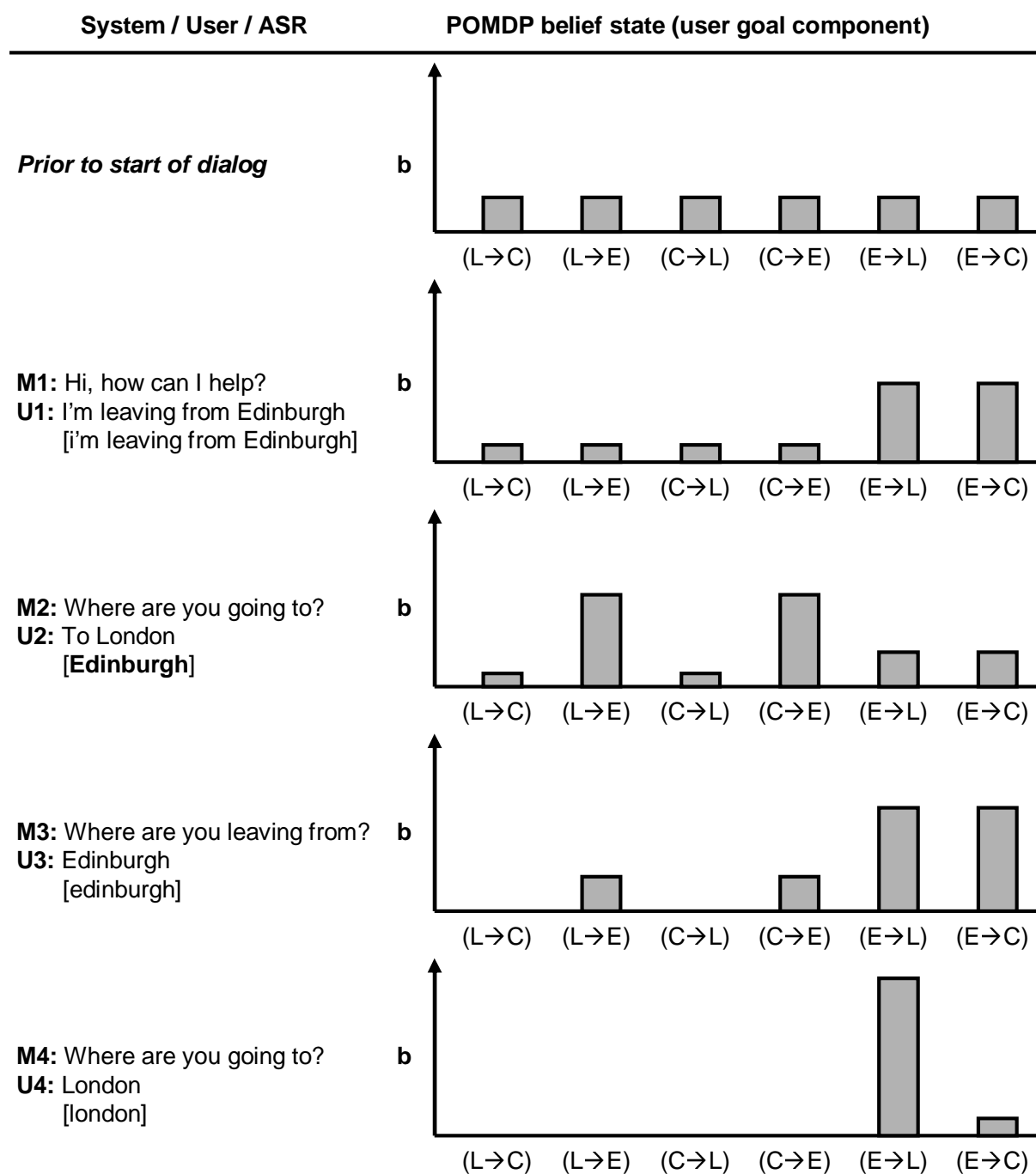


Figure 3.6 Concept error rate (p_{err}) vs. average return (solid line, left axis) and dialog length (dotted line, right axis) for the 3-city TRAVEL SDS-POMDP. Error bars show 95% confidence interval for true average dialog length. As speech recognition errors become more prevalent, average return per dialog decreases and average dialog length increases.



M5: [prints ticket from Edinburgh to London]

Figure 3.7 Example conversation between the user and POMDP dialog controller for the 3-city TRAVEL POMDP example. Graphs show belief mass distribution over the 6 possible user goals s_u ; belief over dialog history s_d and user action a_u are not shown. Cities are abbreviated as (L)ondon, (C)ambridge, and (E)dinburgh, and in this dialog the user wants to travel from Edinburgh to London. A mis-recognition is made when recognizing U2.

Comparisons with existing techniques

The SDS-POMDP model is not the first dialog management framework to address the uncertainty introduced by speech recognition errors, and existing approaches can broadly be grouped into five areas. First, systems can attempt to identify errors locally using a *confidence score*: when a recognition hypothesis has a low confidence score, it can be ignored to reduce the risk of entering bad information into the dialog state. The confidence score itself is unreliable of course, and inevitably bad information will be entered into the dialog state maintained by the system. In view of this, it seems unwise to maintain just one hypothesis for the current dialog state, and a second approach seeks to add robustness by maintaining *parallel state hypotheses*. Dialog is a temporal process in which actions can have long-term consequences which can be difficult for human designers to anticipate. To address this, a third approach performs *automated planning* to find actions which are most useful in the long run.

This chapter first reviews these three techniques – automated planning in section 4.1, confidence scoring in section 4.2, and parallel state hypotheses in section 4.3 – and compares them to the SDS-POMDP model. In each case, it will be shown that the SDS-POMDP model provides an equivalent solution but in a more principled way which admits global parameter optimisation from data. Indeed, it will be shown that each of these existing techniques represents a simplification or special case of the SDS-POMDP model.

Fourth, dispensing with automation, it will then be shown in section 4.4 how SDS-POMDP dialog managers can be compared to hand-crafted dialog managers, and that SDS-POMDP-based dialog managers outperform typical handcrafted baselines. Finally, fifth, in section 4.5 the SDS-POMDP model will be compared to earlier applications of POMDPs to spoken dialog systems, and it will be shown that the SDS-POMDP model represents an advance over past work.

To facilitate comparisons, many dialog management techniques will be shown as influence diagrams, and the SDS-POMDP model itself is shown in Figure 4.1. For clarity Figure 4.1 combines the user's goal s_u , the user's action a_u , and the user's view of the dialog history s_d into one variable $s_m = (s_u, a_u, s_d)$. As is customary for influence diagrams [49], *circles* represent (either discrete or continuous) random variables (also called chance nodes), *squares* represent actions (also called decision nodes), and *diamonds* represent reward nodes (also called utility nodes);

shaded circles indicate unobserved random variables, and *un-shaded* circles represent observed variables; solid directed arcs into random variables indicate causal effect and solid directed arcs into decision nodes represent information passage.

In this chapter, three additions are made to customary notation. First, a *subscript on a decision node* indicates how actions are chosen. For example, in Figure 4.1 the subscript *RL* indicates that actions are chosen using “Reinforcement Learning” (i.e., to maximize expected discounted sum of rewards). Second, the subscript *DET* on a random variable indicates that the variable is a *deterministic* function of its inputs. Table 4.1 lists all node subscripts used in this chapter. Finally, a *dashed directed arc* indicates a distribution is used and not the actual (unobserved) value. For example, in Figure 4.1, the dashed arc from s_m to a_m indicates that a_m is a function of the belief state b (over s_m) and not the actual, unobserved value of s_m .¹

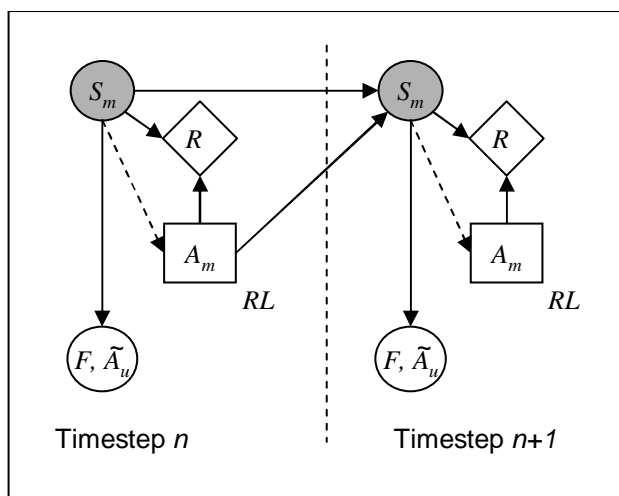


Figure 4.1 SDS-POMDP model shown as an influence diagram. s_m is a composite variable formed of $s_m = (s_u, a_u, s_d)$. The dashed line from s_m to a_m indicates that the action a_m is a function of the belief state $b(s_m)$. The subscript RL indicates that actions are chosen using “Reinforcement Learning”.

Node type	Subscript	Meaning
Random variable (circle)	DET	Deterministic function of inputs
	[none]	Stochastic function of inputs
Action node (square)	HC	Handcrafted
	MEU	Maximum expected utility (max immediate reward)
	RL	Reinforcement Learning (max cumulative reward)
	SL	Supervised Learning

Table 4.1 Node subscripts used in influence diagrams in this chapter.

¹This deviates from typical influence diagram notation in which decision nodes have perfect knowledge of their immediate predecessors. This notation is used to emphasize that decision are taken based on the *belief state* as $a_m = \pi(b)$, and the belief state is equivalent to histories of actions and observations. If traditional influence diagram notation were adhered to, the dashed arc would not appear and instead a solid arc would connect $(\mathcal{F}, \tilde{A}_u)$ to A_m .

4.1 Automated planning

Choosing which action a_m a spoken dialog system should take in a given situation is a difficult task since it is not always obvious what the long-term effect of each action will be. Hand-crafting dialog strategies can lead to unforeseen dialog situations, requiring expensive iterative testing to build good systems. Such problems have prompted researchers to investigate techniques for choosing actions automatically and in this section, the two main approaches to automatic action selection will be considered: supervised learning, and fully-observable Markov decision processes.

As illustrated graphically in Figure 4.2, supervised learning attempts to estimate a direct mapping from machine state s_m to action a_m given a corpus of training examples. In this figure, the decision node a_c indicates the action taken based on the recognition features, for example, to accept or reject the recognition hypothesis.²

Using supervised learning for dialog management in this way can be thought of as a simplification of the SDS-POMDP model in which a single state is maintained, and in which actions are learnt from a corpus. Setting aside the limitations of maintaining just one dialog state and the lack of explicit forward planning, using supervised learning to create a dialog policy is problematic since collecting a suitable training corpus is very difficult for three reasons. Firstly, using human-human conversation data is not appropriate because it does not contain the same distribution of understanding errors, and because human-human turn-taking is much richer than human-machine dialog. As a result, human-machine dialog exhibits very different traits than human-human dialog [26, 70]. Secondly, while it would be possible to use a corpus collected from an existing spoken dialog system, supervised learning would simply learn to approximate the policy used by that spoken dialog system and an overall performance improvement would therefore be unlikely. Thirdly, a corpus could be collected for the purpose, for example, by running Wizard-of-Oz style dialogs in which the wizard is required to select from a list of possible actions at each step [13, 56] or encouraged to pursue more free-form interactions [106, 125]. However, in general such collections are very costly, and tend to be orders of magnitude too small to support robust estimation of generalized action selection.

Fully-observable Markov decision processes (usually just called Markov decision processes, or MDPs) take a very different approach to automated action selection. As their name implies, a Markov decision process is a simplification of a POMDP in which the state is fully observable. This simplification is shown graphically in Figure 4.3. In an MDP, \tilde{a}'_u is again regarded as a random observed variable and s'_m is a deterministic function of s_m , a_m , \tilde{a}'_u , and a'_c . Since at a given state s_m a host of possible observations \tilde{a}'_u are possible, planning is performed using a transition function – i.e. $P(s'_m|s_m, a_m)$. Like POMDPs, MDPs choose actions to maximize a long-term cumulative sum of rewards: i.e., they perform planning. Unlike POMDPs, the current state in an MDP is known, so a policy is expressed directly as a function of state s ; i.e., $\pi : \mathcal{S} \rightarrow \mathcal{A}$. This representation is discrete (a mapping from discrete states to discrete actions), and as a result,

²Handling recognition features in this way is discussed in detail in the next section, starting on page 48.

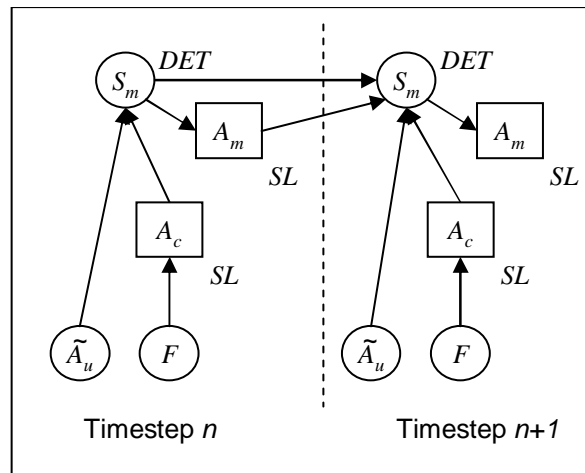


Figure 4.2 *Supervised learning for action selection. The node a_m has been trained using supervised learning on a corpus of dialogs (indicated with the SL subscript). The node a_c indicates what action should be taken based on the recognition features – for example, to accept or reject the recognition hypothesis.*

MDPs are usually regarded as a more tractable formalism than POMDPs. Indeed, MDPs enjoy a rich literature of well-understood optimization techniques and have been applied to numerous real-world problems [88].

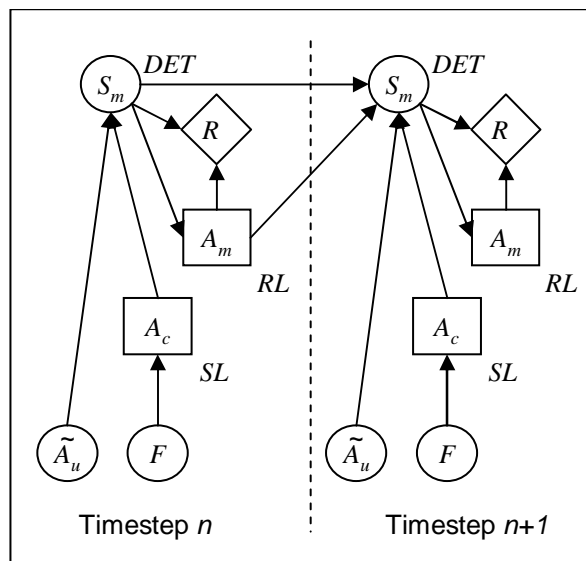


Figure 4.3 *Depiction of an MDP used for dialog management. The action a_m is chosen to maximize the sum of rewards R over time. The node a_c indicates what action should be taken based on the recognition features – for example, to accept or reject the recognition hypothesis.*

By allowing designers to specify rewards for desired and undesired outcomes (e.g., successfully completing a task, a caller hanging up, etc) without specifying explicitly how to achieve each required goal, much of the tedious “handcrafting” of dialog design is avoided. Moreover, unlike the supervised learning approach to action selection, MDPs make principled decisions about the long-term effects of actions, and the value of this approach has been demonstrated

in a number of research systems. For example, in the ATIS Air Travel domain, Levin et al. constructed a system to optimize the costs of querying the user to restrict (or broaden) their flight search, the costs of presenting too many (or too few) flight options, and the costs of accessing a database [60, 61, 62]. In addition, researchers have sought to find optimal initiative, information presentation, and confirmation styles in real dialog systems [105, 115]. MDP-based spoken dialog systems have also given rise to a host of work in user modelling and novel training/optimization techniques [24, 34, 39, 84, 83, 99].

A key weakness of MDPs is that they assume that the current state of the world is known exactly and this assumption is completely unfounded in the presence of recognition errors. The impact of this becomes clear when the MDP transition function is calculated:

$$P(s'_m | s_m, a_m) = \sum_{\tilde{a}'_u} P(\tilde{a}'_u | s_m, a_m) P(s'_m | s_m, a_m, \tilde{a}'_u) \quad (4.1)$$

To compute the transition function properly, an estimate of $P(\tilde{a}'_u | s_m, a_m)$ is required, but in reality \tilde{a}'_u depends critically on hidden variables a_u and s_u . Dialog designers try to ensure that s_m closely models s_u , but as errors are introduced and the two models diverge, the effects of the dependence of \tilde{a}'_u on a hidden variable increasingly violate the Markov assumption expressed in $P(s'_m | s_m, a_m)$, compromising the ability of the MDP to produce good policies. While there exist sophisticated learning techniques (such as eligibility traces) which attempt to compensate for this [99], theory predicts that, as speech recognition errors become more prevalent, POMDPs will outperform MDPs by an increasing margin.

Illustration

To illustrate the advantages of a POMDP over an MDP, consider a spoken dialog system with no confidence scoring and which makes speech recognition errors with a fixed error rate. For this example, which is in the pizza ordering domain, it is assumed that all cooperative user actions are equally likely: i.e., there is no effect of a user model. An example conversation with such a system is shown in Figure 4.4. In this figure, the first column shows interactions between the user and the machine. Text in brackets shows the recognized text (i.e., \tilde{a}'_u). The middle column shows a portion of a POMDP representation of the user's goal. The last column shows how a traditional discrete dialog model like those used in an MDP might track this same portion of the dialog state with a frame-based representation

This conversation illustrates how multiple dialog hypotheses are more robust to errors by properly accounting for conflicting evidence. In this example, the frame-based representation must choose whether to change its value for the *size* field or ignore new evidence; by contrast, the POMDP easily accounts for conflicting evidence by shifting belief mass. Intuitively, a POMDP naturally implements a “best two out of three” strategy.

A POMDP is further improved with the addition of a user model which indicates how a user's goal s_u changes over time, and what actions a_u the user is likely to take in a given situation.

System / User / ASR	POMDP belief state	Traditional method
Prior to start of dialog	<p>A bar chart with a vertical axis labeled 'b' and a horizontal axis with three categories: 'Sml', 'Med', and 'Lrg'. Three bars of equal height are shown, one for each category.</p>	<pre>order: { size: <empty> ... }</pre>
M: How can I help you? U: A small pepperoni pizza [a small pepperoni pizza]	<p>A bar chart with a vertical axis labeled 'b' and a horizontal axis with three categories: 'Sml', 'Med', and 'Lrg'. The 'Sml' bar is significantly taller than the 'Med' and 'Lrg' bars, which are of similar, lower height.</p>	<pre>order: { size: small ... }</pre>
M: Ok, what toppings? U: A small pepperoni [a small pepperoni]	<p>A bar chart with a vertical axis labeled 'b' and a horizontal axis with three categories: 'Sml', 'Med', and 'Lrg'. The 'Sml' bar is significantly taller than the 'Med' and 'Lrg' bars, which are of similar, lower height.</p>	<pre>order: { size: small ... }</pre>
M: And what type of crust? U: Uh just normal [large normal]	<p>A bar chart with a vertical axis labeled 'b' and a horizontal axis with three categories: 'Sml', 'Med', and 'Lrg'. The 'Sml' bar is tall, the 'Med' bar is very short, and the 'Lrg' bar is of medium height.</p>	<pre>order: { size: large [?] ... }</pre>

Figure 4.4 Example conversation with a spoken dialog system illustrating the benefit of maintaining multiple dialog state hypotheses. This example is in the pizza ordering domain. The left column shows the machine and user utterances, and the recognition results from the user’s utterance is shown in brackets. The center column shows a portion of the POMDP belief state; b represents the belief over a component of the user’s goal (pizza size). The right-hand column shows a typical frame-based method which is also tracking this component of the user’s goal. Note that a speech recognition error is made in the last turn – this causes the traditional method to absorb a piece of bad information, whereas the POMDP belief state is more robust.

For example, consider the dialog shown in Figure 4.5. In this figure, a user model informs the likelihood of each recognition hypothesis \tilde{a}'_u given s_u and a_m .

In this example, the machine asks for the value of one slot, and receives a reply. The system then asks for the value of a second slot, and receives a value for that slot and an inconsistent value for the first slot. In the traditional frame-based dialog manager, it is unclear whether the new information should replace the old information or should be ignored. Further if the frame is extended to allow conflicts, it is unclear how they will be resolved, and how the fact that the new evidence is less likely than the initial evidence be reflected. By contrast, in the SDS-POMDP the belief state update is scaled by the likelihood predicted by the user model. In other words, the POMDP takes minimal (but non-zero) account of very unlikely user actions it observes, and

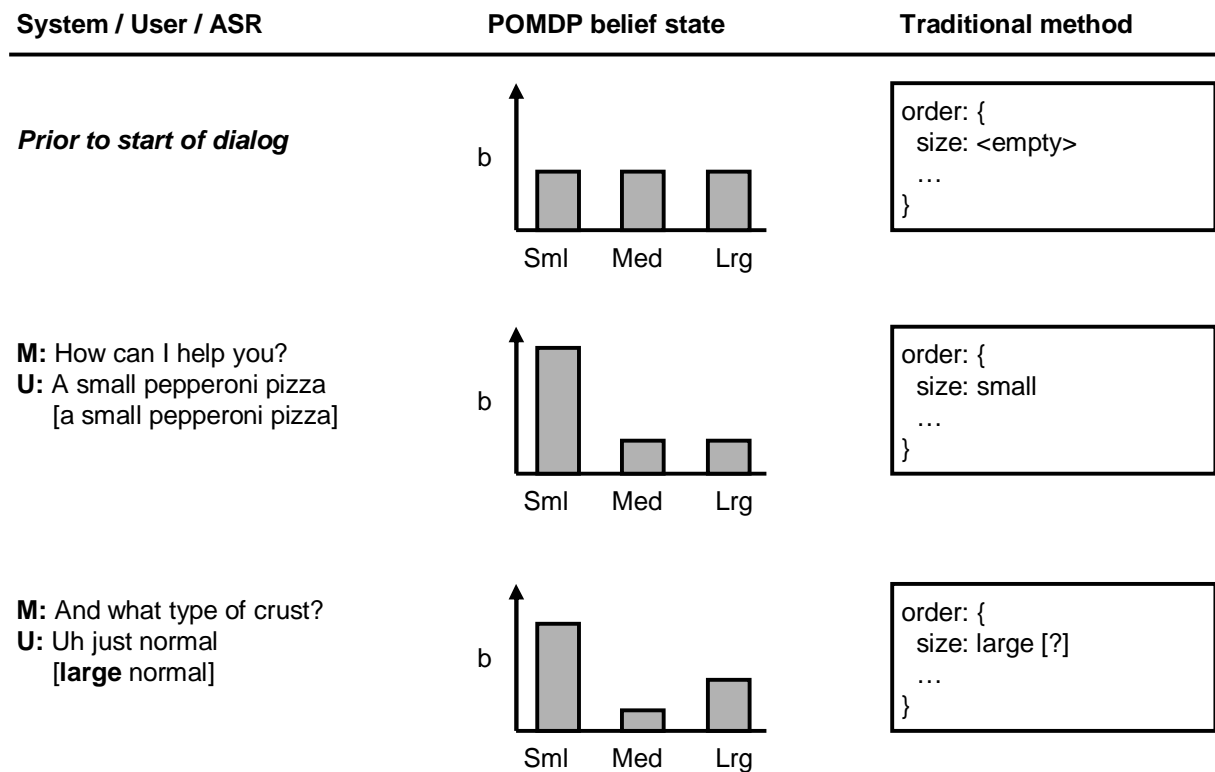


Figure 4.5 Example conversation with a spoken dialog system illustrating the benefit of an embedded user model. In the POMDP, for the first recognition, the observed user’s response is very likely according to the user model. The result is a large shift in belief mass toward the Sml value. In the second recognition, providing information about the size is predicted as being less likely; as a result, the observed response Lrg (a speech recognition error) is given less weight, and the final POMDP belief state has more mass on Sml than Lrg. By contrast, the traditional discrete method must choose whether to update the state with Sml or Lrg.

maximal account of very likely actions it observes.

Results

To compare the SDS-POMDP model to an MDP-based dialog manager, an MDP was constructed for the TRAVEL application, patterned on systems in the literature [83]. In this section, comparisons are made without access to a confidence score (i.e., $h = 0$). The MDP was trained and evaluated through interaction with a model of the environment, which was formed from the POMDP transition, observation, and reward functions. This model of the environment takes an action from the MDP as input, and emits an observation and a reward to the MDP as output.

The MDP state contains components for each field which reflect whether, from the standpoint of the machine, a value has not been observed, a value has been observed but not confirmed, or a value has been confirmed. Two additional states – dialog-start and dialog-end – which were also in the POMDP state space, are included in the MDP state space for a total of 11 MDP states, shown in Table 4.2. The MDP was optimized using Watkins Q-Learning [121].

Figure 4.6 shows the average return (i.e. total cumulative reward) for the POMDP and MDP

Listing of MDP states		
u-u	o-u	c-u
u-o	o-o	c-o
u-c	o-c	c-c
dialog-start dialog-end		

Table 4.2 *The 11 MDP states used in the test-bed simulation. In the items of the form x-y, the first element x refers to the from slot, and the second element y refers to the to slot. u indicates unknown; o indicates observed but not confirmed; c indicates confirmed.*

solutions vs. the concept error rate (p_{err}) ranging from 0.00 to 0.65. The (negligible) error bars for the MDP show the 95% confidence interval for the estimate of the return assuming a normal distribution. The POMDP and MDP perform equivalently when no recognition errors are made ($p_{err} = 0$), and the return for both methods decreases consistently as the concept error rate increases but the POMDP solution consistently achieves the larger return. Thus, in the presence of perfect recognition accuracy, there is no advantage to maintaining multiple dialog states, but when errors do occur the POMDP solution is always better, and furthermore the difference in performance increases as the concept error rate increases. This result confirms that the use of multiple dialog hypotheses and an embedded user model enable higher recognition error rates to be tolerated compared to the conventional single-state approach. A detailed inspection of the dialog transcripts confirmed that the POMDP is better at interpreting inconsistent information, agreeing with the intuition shown in Figure 4.4.

Although there is little related work in the literature in this area, these findings agree with past experiments which have also showed performance gains of POMDPs (or approximations of POMDPs) over MDPs in this domain [92, 132, 133]. These works are described below in section 4.5 (page 60).

4.2 Local confidence scores

Most speech recognition engines annotate their output word hypotheses \tilde{w} with confidence scores $P(\tilde{w}|y_u)$ and in some domains, modern systems can compute this measure accurately [30, 53, 71]. Subsequent processing in the speech understanding components will often augment this low level acoustic confidence using extra features such as parse scores, prosodic features, dialog state, etc [80, 54, 10, 32]. Schemes have also been proposed to identify utterances which are likely to be problematic or corrections [64, 55, 66, 65, 42, 41], or to predict when a dialog is likely to fail [57].

For the purposes of a dialog system, however, the essential point of a confidence score is that it provides a mapping from a set of features \mathcal{F} into a single metric c which yields an overall indication of the reliability of the hypothesized user intention \tilde{a}_u . Traditional systems typically incorporate confidence scores by specifying a confidence threshold c_{thresh} which implements an accept/reject decision for an \tilde{a}_u : if $c > c_{thresh}$ then \tilde{a}_u is deemed reliable and accepted;

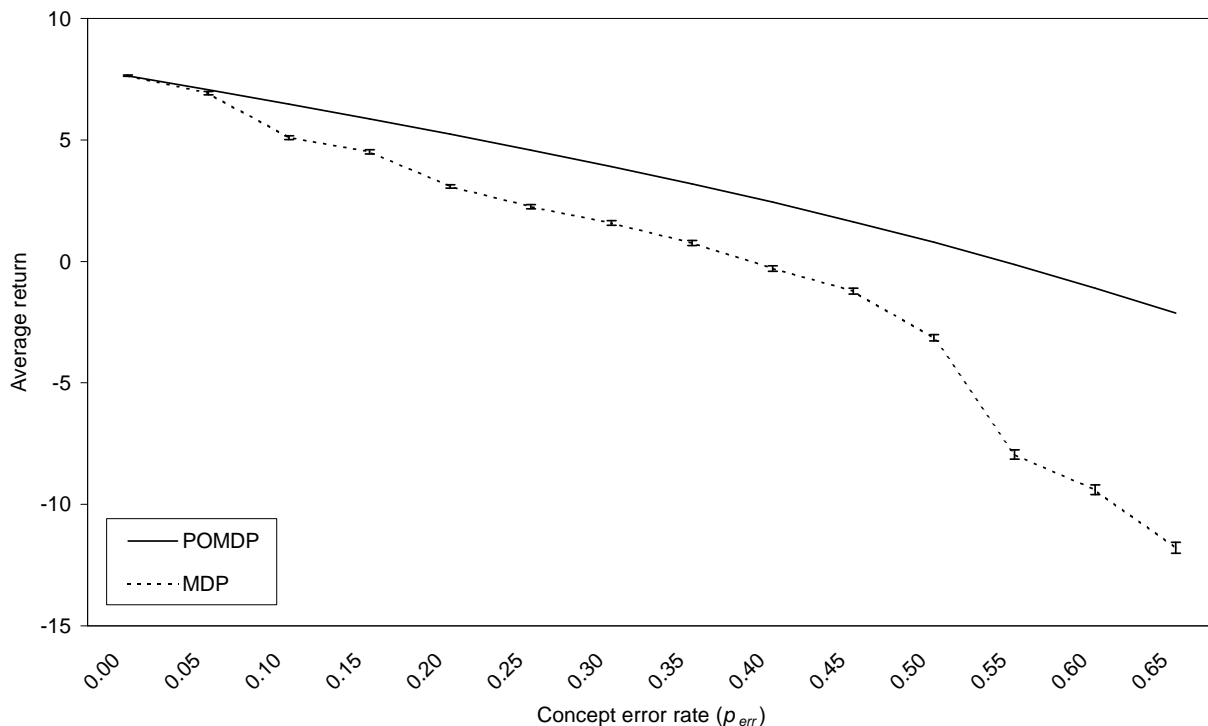


Figure 4.6 Concept error rate vs. average return for the POMDP policy and the MDP baseline. Error bars show the 95% confidence interval.

otherwise it is deemed unreliable and discarded. In practice any value of c_{thresh} will still result in classification errors, so c_{thresh} can be viewed as implementing a trade-off between the cost of a false-negative (rejecting an accurate \tilde{a}_u) and the cost of a false-positive (accepting an erroneous \tilde{a}_u).

Figure 4.7 shows how a spoken dialog system with a confidence score can be expressed in an influence diagram. a_c is a decision node that indicates the “confidence bucket” – for example, $\{hi, low, reject\}$. a_c is typically trained using a corpus of examples and supervised learning, indicated by the subscript SL on the node a_c . This “confidence bucket” is then incorporated into the dialog state using hand-crafted update rules of the form $s'_m = f(s_m, a_m, a'_c, \tilde{a}'_u)$.³ Based on the updated dialog state s_m , the policy determines which action to take. Typically actions are selected with handcrafted rules, as shown by the subscript HC in Figure 4.7.

Figure 4.7 also highlights key differences between a traditional system with a confidence score and the SDS-POMDP model. In both models, \tilde{a}_u and f are regarded as observed random variables. However, in the confidence score approach, a hard and coarse decision is made about the validity of \tilde{a}_u via the decision a_c . The decision implemented in a_c is non-trivial since there is no principled way of setting the confidence threshold c_{thresh} . In practice a developer will look at expected accept/reject figures and use intuition. A slightly more principled approach would attempt to assign costs to various outcomes (e.g., cost of a false-accept, cost of a false reject, etc.)

³As above, the superscript DET on the node s_m indicates that s_m takes on a deterministic value: for a known set of inputs, it yields exactly one output.

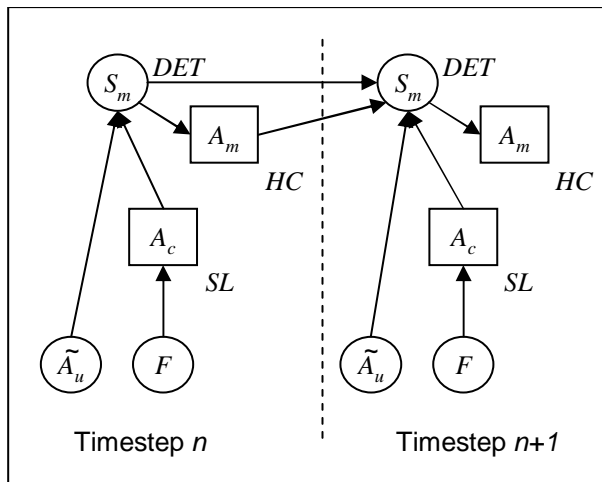


Figure 4.7 Influence diagram showing how a confidence score is typically incorporated into a spoken dialog system. Node \mathcal{F} is a random variable incorporating a set of recognition features such as likelihood ratio, hypothesis density, parse coverage, etc. A_c includes actions such as {hi, low, reject}.

and choose a threshold accordingly [9, 78]. However, these costs are specified in immediate terms, whereas in practice the decisions have long-term effects (e.g., subsequent corrections) which are difficult to quantify, and which vary depending on context. Indeed, when long-term costs are properly considered, there is evidence that values for optimal confidence thresholds are not at all intuitive: one recent study found that for many interactions, the optimal confidence threshold was zero – i.e., any recognition hypothesis, no matter how poorly scored, should be accepted [12].

By contrast, the SDS-POMDP approach models the recognition features themselves as continuous observed random variables. Note how in Figure 4.7, the recognition features are viewed as functional inputs, whereas in the POMDP (Figure 4.1), they are viewed as observed outputs from a hidden variable. In this way, the SDS-POMDP never makes hard accept/reject decisions about evidence it receives, but rather uses recognition features to perform inference over all possible user actions a_u . Further, the explicit machine dialog state s_m used in traditional approaches is challenged to maintain a meaningful confidence score history since typically if a value of \tilde{a}'_u is rejected, that information is discarded. By contrast, the SDS-POMDP aggregates all information over time including conflicting evidence via a belief state, properly accounting for the reliability of each observation in cumulative terms. Finally, whereas accept/reject decisions in a traditional system are taken based on local notions (often human intuitions) of utility, in the SDS-POMDP actions are selected based on expected long-term reward – note how Figure 4.1 explicitly includes a reward component, absent from Figure 4.7.

Incorporating a confidence score into a traditional hand-crafted SDS does add useful information, but acting on this information in a way which serves long-term goals is non-trivial. A traditional SDS with a confidence score can be viewed as an SDS-POMDP with a number of simplifications: one dialog state is maintained rather than many; accept/reject decisions are used

in place of parallel dialog hypotheses; and actions are selected based on a hand-crafted strategy rather than selected to maximize a long-term reward metric.

Illustration

To illustrate the difference between a traditional implementation of a confidence score in a dialog system and the SDS-POMDP approach, consider a spoken dialog system which makes use of a per-utterance confidence score which ranges from 0 to 1. Assume that all cooperative user actions are equally likely so that the effects of a user model can be disregarded. In the traditional version of this system with three confidence buckets $\{hi, low, reject\}$, suppose that a good threshold between *reject* and *low* has been found to be 0.4, and a good threshold between *low* and *hi* has been found to be 0.8.

An example conversation is shown in Figure 4.8 in which the machine asks a question and correctly recognizes the response. In the traditional method, the confidence score of 0.85 is in the *hi* confidence bucket, hence the utterance is accepted and the dialog state is updated accordingly. In the POMDP, the confidence score is incorporated into the magnitude of the belief state update.

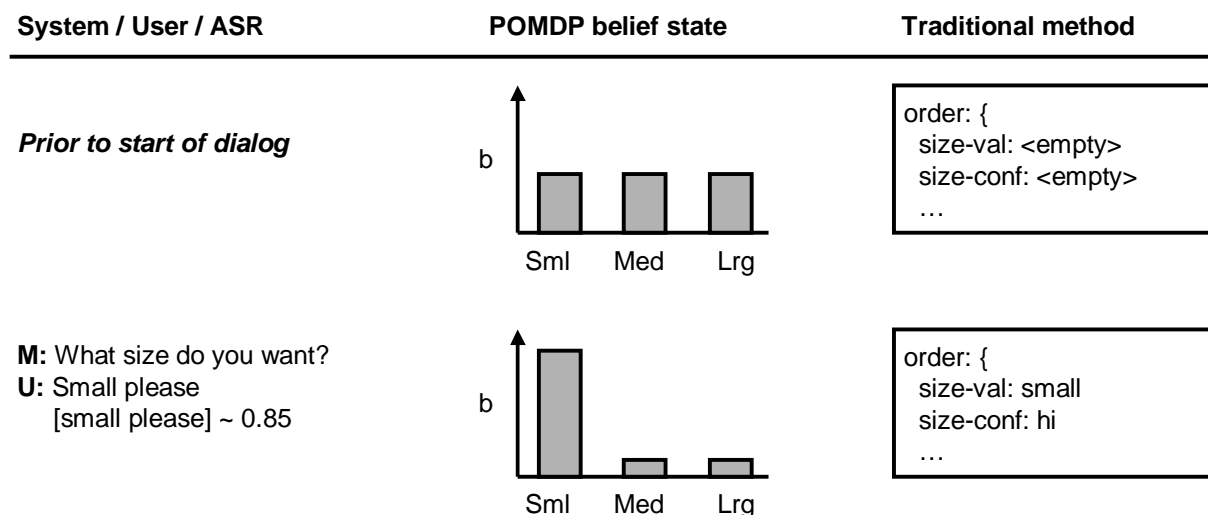


Figure 4.8 Example conversation with a spoken dialog system illustrating a high-confidence recognition. The POMDP incorporates the magnitude of the confidence score by scaling the belief state update, and the traditional method quantizes the confidence score into a “bucket” such as $\{hi, low, reject\}$.

By contrast, consider the conversation in Figure 4.9, in which each of the recognitions is again correct, but the confidence scores are lower. In the traditional method, each confidence score falls into the *reject* confidence bucket, and nothing is incorporated into the dialog frame. In the POMDP-based system, however, the magnitude of the confidence score is incorporated into the belief update as above, although this time since the score is lower, each update shifts less belief mass.

This second example illustrates two key benefits of using POMDPs for dialog management.

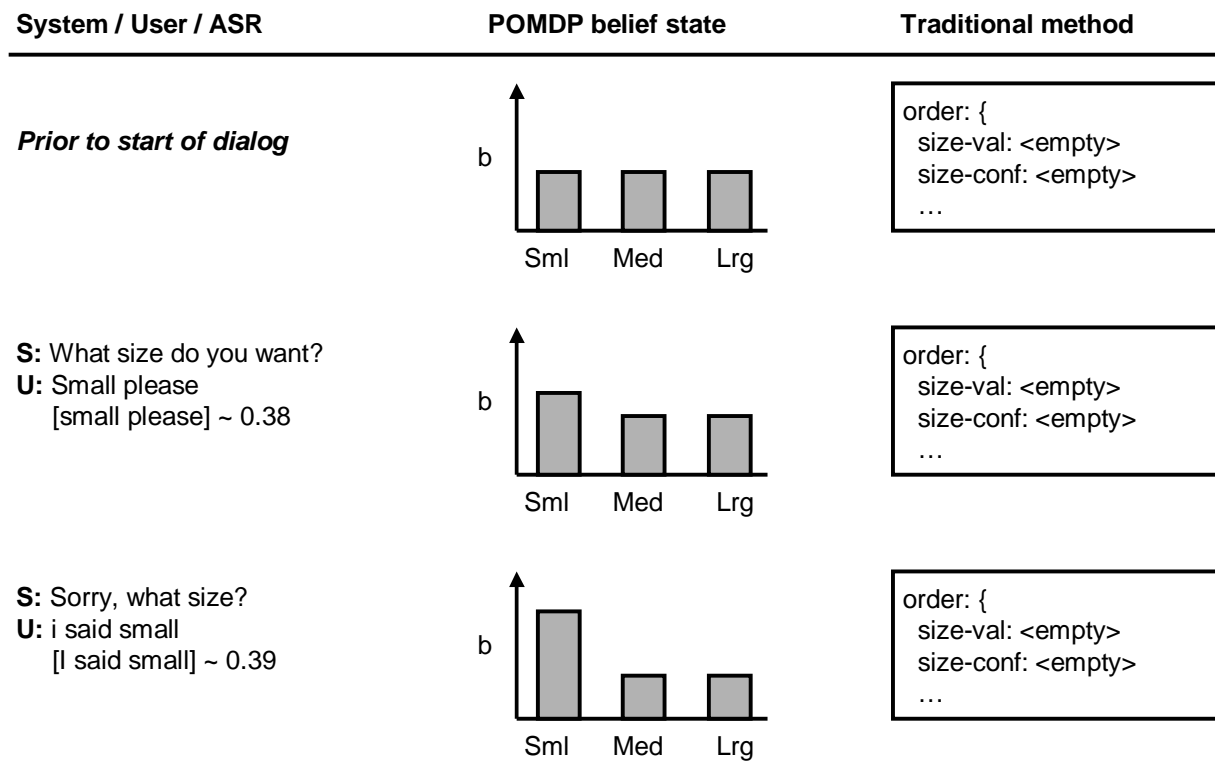


Figure 4.9 Example conversation with a spoken dialog system illustrating two successive low-confidence recognitions. In this example, both recognitions are correct. The POMDP accumulates weak evidence over time, but the traditional method ignores both recognitions because they are below the threshold of 0.40. In effect, the traditional method is discarding possibly useful information.

First, looking *within* one time-step, whereas the traditional method creates a finite set of confidence buckets, the POMDP in effect utilizes an infinite number of confidence buckets and as a result the POMDP belief state is a lossless representation of the information conveyed in the confidence score within one time-step. Second, looking *across* time-steps, whereas the traditional method is challenged to track aggregate evidence about confidence scores over time, a POMDP effectively maintains a cumulative confidence score over user goals. For the traditional method to approximate a cumulative confidence score, a policy which acted on a historical record of confidence scores would need to be devised, and it is unclear how to do this.

Moreover, the incorporation of confidence score information and user model information are complementary since they are separate product terms in the belief update (Equation 3.11). The probability $P(\tilde{a}'_u, c' | a'_u)$ reflects the contribution of the confidence score and the probability $P(a'_u | s'_u, a_m)$ reflects the contribution of the user model.⁴ The belief term $b(s_u, a_u, s_d)$ records the dialog history and provides the memory needed to accumulate evidence. This is in contrast to traditional approaches which typically have a small number of confidence score “buckets” for each recognition event, and typically log only the most recently observed “bucket”. POMDPs have in effect infinitely many confidence score buckets *and* they aggregate evidence properly over time as a well-formed distribution over all possible dialog states (including user goals).

⁴Since confidence score c is the recognition feature f considered here, $c = f$.

Results

To test these intuitions experimentally, the TRAVEL dialog management problem was assessed using a confidence score (i.e., with $h > 0$) by incorporating confidence score information into the belief monitoring process [123].⁵ The MDP baseline was extended to include M confidence buckets, patterned on systems in the literature [83]. Ideally the thresholds between confidence buckets would be selected so that they maximize average return; however, it is not obvious how to perform this selection – indeed, this is one of the weaknesses of “confidence bucket” method. Instead, a variety of techniques for setting confidence score threshold were explored, and it was found that dividing the probability mass of the confidence score c evenly between buckets produced the largest average returns.

The MDP state was extended to include this confidence “bucket” information. Because the confidence bucket for each field (including its value and its confirmation) is tracked in the MDP state, the size of the MDP state space grows with the number of confidence buckets. For $M = 2$, the resulting MDP, called “MDP-2”, has 51 states; this is shown in Table 4.3. Watkins Q-learning was again used for MDP optimization.

List of MDP-2 states						
u-u	u-o(l)	u-o(h)	u-c(l,l)	u-c(l,h)	u-c(h,l)	u-c(h,h)
o(l)-u	o(l)-o(l)	o(l)-o(h)	o(l)-c(l,l)	o(l)-c(l,h)	o(l)-c(h,l)	o(l)-c(h,h)
o(h)-u	o(h)-o(l)	o(h)-o(h)	o(h)-c(l,l)	o(h)-c(l,h)	o(h)-c(h,l)	o(h)-c(h,h)
c(l,l)-u	c(l,l)-o(l)	c(l,l)-o(h)	c(l,l)-c(l,l)	c(l,l)-c(l,h)	c(l,l)-c(h,l)	c(l,l)-c(h,h)
c(l,h)-u	c(l,h)-o(l)	c(l,h)-o(h)	c(l,h)-c(l,l)	c(l,h)-c(l,h)	c(l,h)-c(h,l)	c(l,h)-c(h,h)
c(h,l)-u	c(h,l)-o(l)	c(h,l)-o(h)	c(h,l)-c(l,l)	c(h,l)-c(l,h)	c(h,l)-c(h,l)	c(h,l)-c(h,h)
c(h,h)-u	c(h,h)-o(l)	c(h,h)-o(h)	c(h,h)-c(l,l)	c(h,h)-c(l,h)	c(h,h)-c(h,l)	c(h,h)-c(h,h)
dialog-start dialog-end						

Table 4.3 *The 51 states in the “MDP-2” simulation. In the items of the form x-y, the first element x refers to the from slot, and the second element y refers to the to slot. u indicates unknown; o indicates observed but not confirmed; c indicates confirmed. o(l) means that the value was observed with low confidence; o(h) means that the value was observed with high confidence. c(l,l) means that both the value itself and the confirmation were observed with low confidence; c(l,h) means that the value was observed with low confidence and the confirmation was observed with high confidence, etc.*

Figure 4.10 shows concept error rate vs. average returns for the POMDP and MDP-2 solutions for $h = 1$. The error bars show the 95% confidence intervals for the return assuming a normal distribution. Return decreases consistently as the concept error rate increases for all solution

⁵The POMDP dialog controller was the same as that used in the previous section, 4.1, and did not take account of the confidence score when *planning*. In other words, confidence score was used to more accurately estimate the distribution over user goals given the dialog history, but actions were chosen with the expectation that, *in the future*, confidence score information would not be available. Other experiments were performed in which optimization was performed *with* confidence score information using a technique that admits continuous observations [43], and this did not improve performance significantly [123].

methods. When no recognition error are made ($p_{err} = 0$), the two methods perform identically; when errors are made ($p_{err} > 0$), the POMDP solutions attain larger returns than the MDP method by a significant margin.

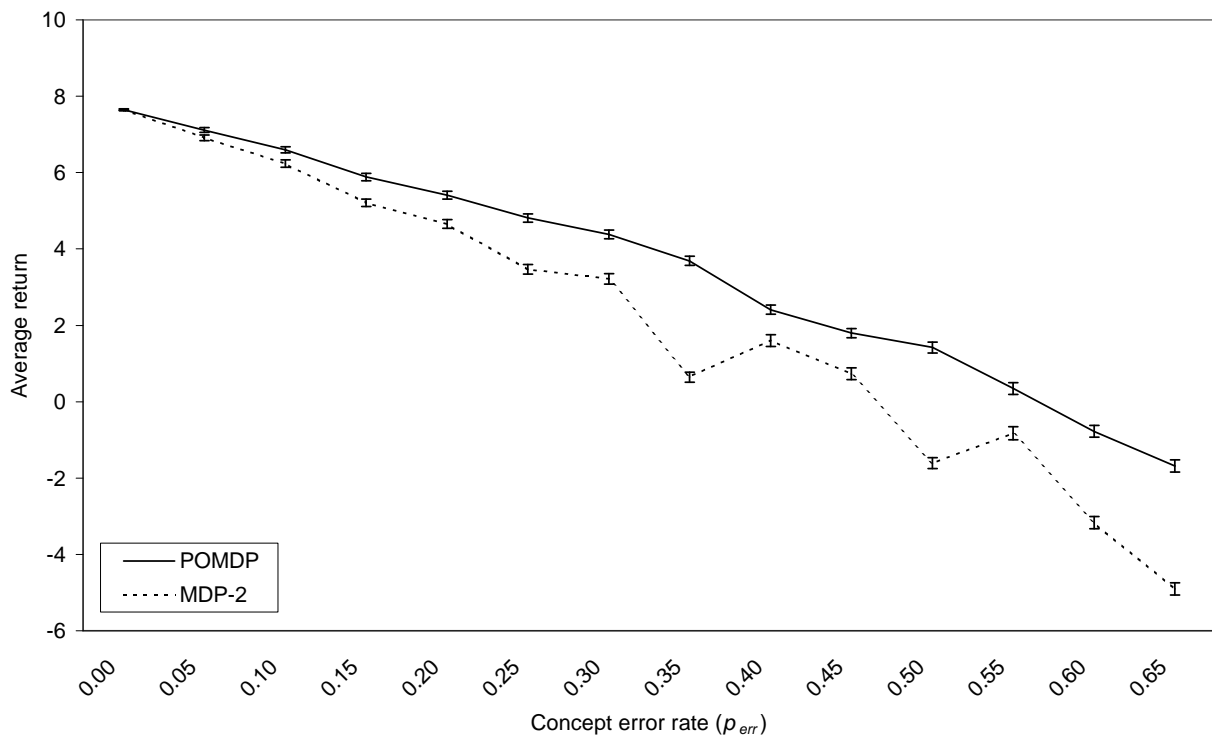


Figure 4.10 Concept error rate (p_{err}) vs. average return for the POMDP and MDP-2 baseline for $h = 1$.

Next, the effects of varying the informativeness of the confidence score were explored. Figure 4.11 shows the informativeness of the confidence score (h) vs. average returns for the POMDP method and the MDP-2 method when the concept error rate (p_{err}) is 0.3. The error bars show the 95% confidence interval for return assuming a normal distribution. Increasing h increases average return for both methods, but the POMDP method consistently outperforms the baseline MDP method. This trend was also observed for a range of other concept error rates [123].

In this example, confidence score was considered as a component of the POMDP observation. Equally the recognition features which make up confidence score could be used as evidence directly – in fact, any recognition feature for which a model can be constructed can be incorporated as evidence and this has been demonstrated in for example [44].

4.3 Parallel state hypotheses

Traditional dialog management schemes maintain (exactly) one dialog state $s_m \in \mathcal{S}_m$, and when a recognition error is made, s_m may contain erroneous information. Although designers have developed *ad hoc* techniques to avoid dialog breakdowns such as allowing a user to “undo” system mistakes, the desire for an inherently robust approach remains. A natural approach to coping with erroneous evidence is to maintain multiple hypotheses for the correct dialog state.

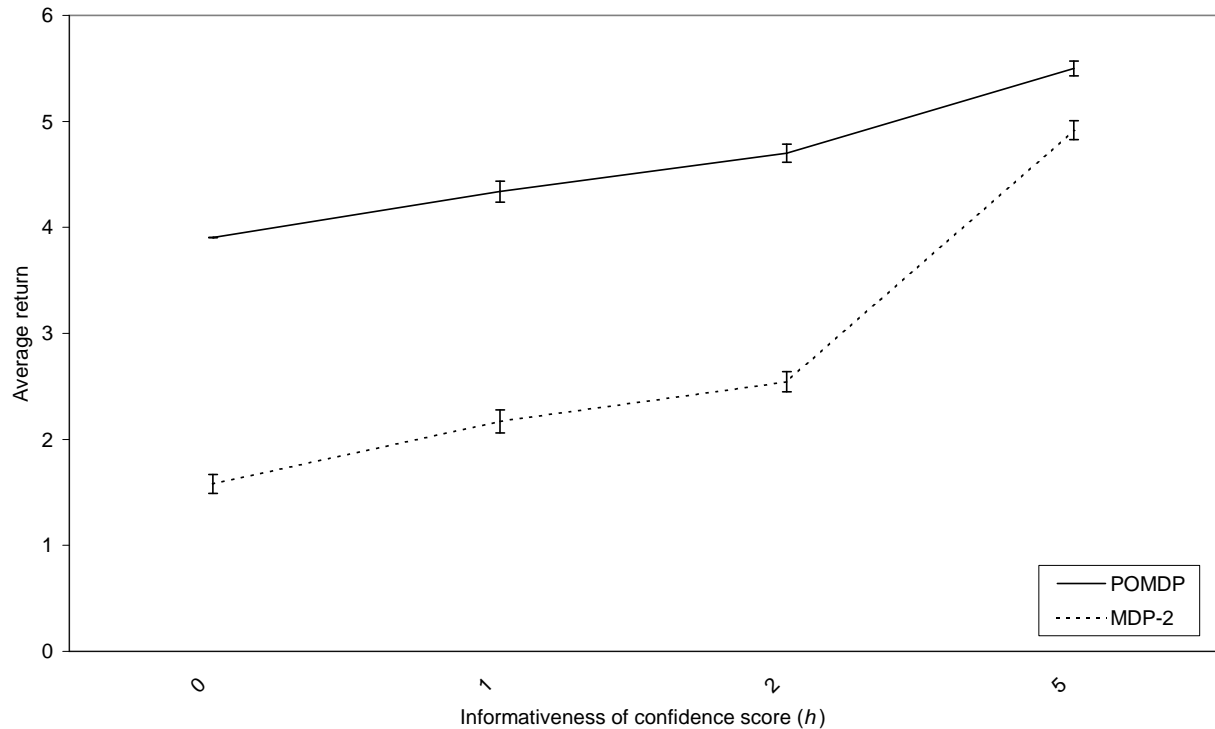


Figure 4.11 Confidence score informativeness (h) vs. average return for the POMDP and MDP-2 baseline for a concept error rate (p_{err}) of 0.30.

Similar to a beam search in a hidden Markov model, tracking multiple dialog hypotheses allows a system to maintain different explanations for the evidence received, always allowing for the possibility that each individual piece of evidence is an error. In this section, two techniques for maintaining multiple dialog hypotheses are reviewed: greedy decision theoretic approaches and an M-Best list.

Greedy decision theoretic approaches construct an influence diagram as shown in Figure 4.12 [76, 77]. The structure of the network is identical to a POMDP: the machine maintains a belief state over a hidden variable s_m , and s_m may be factored as in SDS-POMDP model. The dashed line in the figure from s_m to a_m indicates that a_m is chosen based on the distribution over s_m rather than its actual (unobserved) value. As with a POMDP, a reward (also called a utility) function is used to select actions – however, greedy decision theoretic approaches differ from a POMDP in *how* the reward is used to selection actions. Unlike a POMDP, in which machine actions are chosen to maximize the cumulative long-term reward, greedy decision theoretic approaches choose the action which maximizes the immediate reward. In other words, greedy decision theoretic approaches can be viewed as a special case of a POMDP in which planning is not performed; i.e., a POMDP with planning horizon zero, or equivalently, a POMDP with discount $\gamma = 0$. As such, action selection is tractable for real-world dialog problems, and greedy decision theoretic approaches have been successfully demonstrated in real working dialog systems [44, 79].

Whether the dialog manager explicitly performs planning or not, a successful dialog must

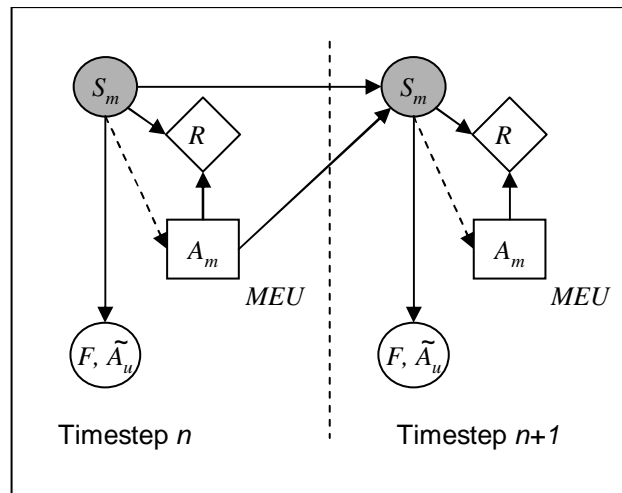


Figure 4.12 View of a spoken dialog system as a greedy decision theoretic process. Action a_m is selected to maximize the expected immediately utility r , indicated by the subscript MEU (“Maximum Expected Utility”). The dashed line indicates that a_m is a function of the distribution over s_m , rather than its actual (unobserved) value.

make progress to some long-term goal. In greedy decision theoretic approaches, a system will make long-term progress toward a goal only if the reward metric has been carefully crafted. Unfortunately, crafting a reward measure which accomplishes this is a non-trivial problem and in practice encouraging a system to make progress to long-term goals inevitably requires some hand-crafting resulting in the need for *ad hoc* iterative tuning.

An alternative to the greedy decision theoretic approach is to still maintain multiple dialog hypotheses but select actions by considering only the *top* dialog hypothesis, using a handcrafted policy as in conventional heuristic SDS design practice. This approach is referred to as the M-Best list approximation, and it is shown graphically in Figure 4.13. In this figure, the subscript DET indicates that the node s_m^* is not random but rather takes on a deterministic value for known inputs, and here s_m^* is set to the state s_m with the most belief mass. The M-best list approach has been used to build real dialog systems and shown to give performance gains relative to an equivalent single-state system [40].

The M-best approximation can be viewed as a POMDP in which action selection is hand-crafted, and based only on the most likely dialog state. When cast in these terms, it is clear that an M-best approximation makes use of only a fraction of the available state information since considering only the top hypothesis may ignore important information in the alternative hypotheses such as the whether the second-best is very similar or very different to the best hypothesis. Hence, even setting aside the use of *ad hoc* hand-crafted policies, the M-best list approach is clearly sub-optimal. In contrast, since the SDS-POMDP constructs a policy which covers belief space, it considers *all* alternative hypotheses.

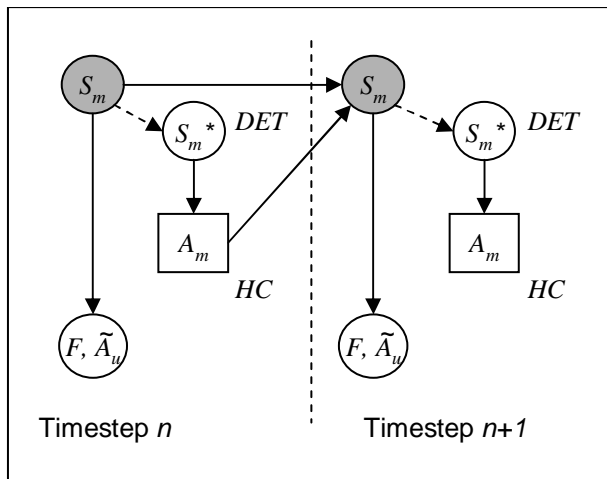


Figure 4.13 Influence diagram showing multiple state hypotheses. s_m^* takes the value of the state s_m with the highest belief mass at each time-step.

Illustration

To illustrate why POMDPs outperform greedy methods, consider the VOICEMAIL application presented in section 2.2 (page 9). Briefly summarized, in this POMDP the machine can *ask* the user whether they would like a message saved or deleted, or can take the *doSave* or *doDelete* actions and move on to the next message. Observations \overline{save} and \overline{delete} give the results from the speech recognition process and may contain errors.

The POMDP policy with horizon $t = 1$ is equivalent to the greedy decision theoretic policy; this is in effect the first iteration of value iteration (shown in Figure 2.6, page 16), and this 1-step policy is shown in the upper section of Figure 4.14. By continuing to run value iteration, a POMDP policy which maximizes cumulative discounted reward is produced (Figure 2.9, page 18), and this produces an (approximation to) infinite-horizon policy, shown in the lower section of Figure 4.14. Note that the POMDP policy is more conservative in that the central region corresponding to the *ask* action is larger. This is a consequence of planning, which has determined that the expected benefit of gathering additional information (and increasing its certainty in the belief state) outweighs the short-term cost of gathering that information. Contrast this to the greedy policy, which takes either the *doSave* or *doDelete* action whenever its expected *immediate* reward is greater than that of the *ask* action.

Results

The empirical benefit of planning is assessed by comparing the performance of a POMDP to a greedy decision theoretic dialog manager on the TRAVEL application. This greedy dialog manager always takes the action with the highest expected immediate reward – i.e., unlike a POMDP, it is not performing planning. Both dialog managers were evaluated by simulating conversations and finding the average reward gained per dialog. Results are shown in Figure 4.15. The POMDP outperforms the greedy method by a large margin for all error rates. Intuitively,

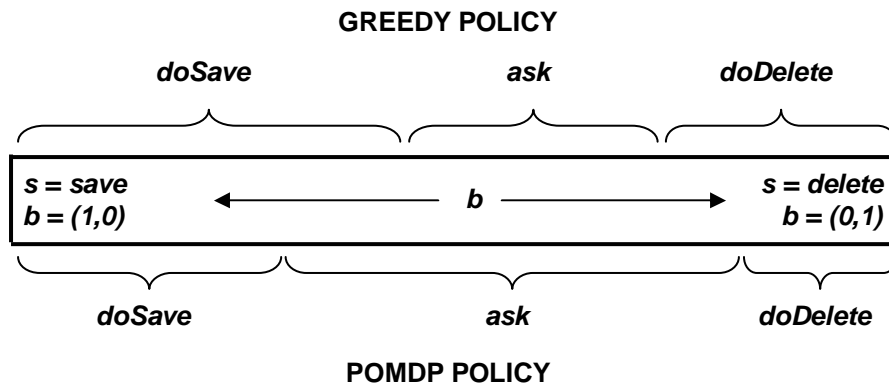


Figure 4.14 Greedy (top) vs. POMDP (bottom) policies for the VOICEMAIL application. Note that the region corresponding to the ask action is larger for the POMDP policy: the POMDP policy has calculated portions to maximize long-term reward, whereas the greedy policy chooses actions to maximize immediate reward.

the POMDP is able to reason about the future and determine when gathering information will reap larger gains in the long term even if it incurs an immediate cost. More specifically, in this example, the POMDP gathers more information than the greedy approach. As a result, dialogs with the POMDP dialog manager are longer but the resulting increased cost is offset by correctly identifying the user’s goal more often. In general, POMDPs are noted for their ability to make effective trade-offs between the (small) cost of gathering information, the (large) cost of acting on incorrect information, and rewards for acting on correct information [17].

4.4 Handcrafted dialog managers

Historically, the design of dialog systems has relied on the intuition and experience of human designers to *handcraft* a mapping from dialog states s_m to system action a_m , requiring the designer to anticipate where uncertainty is likely to arise and to choose actions accordingly. Handcrafting can be a time-consuming process and as a result researchers have proposed a host of higher-level frameworks to help designers specify systems more efficiently [101, 59, 11, 1, 23, 75]. Most commercially deployed applications are handcrafted, drawing on considerable expertise and widespread higher-level specification languages such as VoiceXML [5, 6, 20, 67].

To compare a POMDP policy with a hand-crafted policy, first the form of POMDP policies must be considered. To this point, policies have been represented as max operation over a set of value functions $\{V_T^n(b) : 1 \leq n \leq N\}$, where each value function V_T^n describes the value of a T -step conditional plan n which begins with action a^n . A machine behaves optimally by, at each time-step, determining its belief state b , and taking action a^n where $n = \arg \max_{\tilde{n}} \sum_s b(s) V_T^{\tilde{n}}(s)$. Stated alternatively, a policy is a mapping $\pi : \mathcal{B} \rightarrow \mathcal{A}$ which *partitions* belief space into regions where different conditional plans are optimal.

Handcrafted dialog managers are not specified this way: instead, they are typically specified as a policy graph. A *policy graph* is a finite state controller (FSC) consisting of a set of nodes \mathcal{M} . Each controller node is assigned a POMDP action, where $\pi_{FSC} : \mathcal{M} \rightarrow \mathcal{A}$. Arcs are labelled

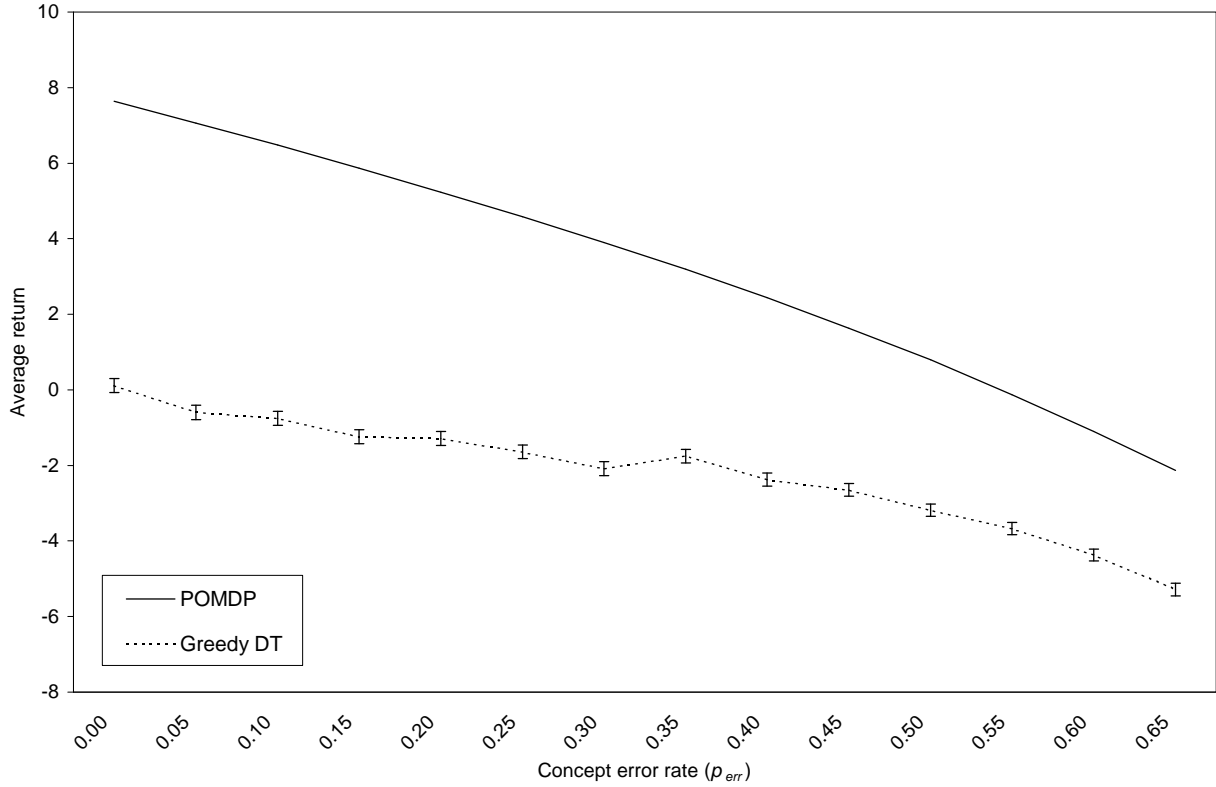


Figure 4.15 Concept error rate (p_{err}) vs. average return for POMDP and greedy decision theoretic (“Greedy DT”) dialog managers.

with a POMDP observation, such that all controller nodes have exactly one outward arc for each observation. $l(m, o')$ denotes the successor node for node m and observation o' . A policy graph is a general and common way of representing handcrafted dialog management policies [82], and rule-based formalisms like those mentioned above can most always be compiled into a (possibly very large) policy graph.

Unlike a policy represented as a collection of value functions, a policy graph does not make the expected return associated with each controller node explicit. However, as pointed out by Hansen [36], the expected return associated with each controller node can be found by solving a system of linear equations in V :

$$V_m(s) = \text{Immediate reward} + \text{discount} \cdot \text{expected future reward} \quad (4.2)$$

$$V_m(s) = r(s, \pi_{FSC}(m)) + \gamma \sum_{s'} \sum_{o'} P(s'|s, \pi_{FSC}(m)) P(o'|s', \pi_{FSC}(m)) V_{l(m, o')}(s'). \quad (4.3)$$

Solving this set of linear equations yields a set of vectors with one vector for each controller node. The expected value of starting the controller in node m and belief state b can be computed by evaluating $\sum_s V_m(s)b(s)$. For a given belief state b , the controller node m^* which maximizes expected return is

$$m^* = \arg \max_m \sum_s V_m(s)b(s) \quad (4.4)$$

Like an MDP, a handcrafted dialog manager maintains one dialog state, and thus suffers from the same limitations as shown in the illustrative dialogs in section 4.1 (page 45).

Results

To illustrate policy graph evaluation, three handcrafted policies called HC1, HC2, and HC3 were created for the TRAVEL application. Each of these policies encode strategies typically used by designers of spoken dialog systems. All of the handcrafted policies first take the action *greet*. HC1 takes the *ask-from* and *ask-to* actions to fill the *from* and *to* fields, performing no confirmation. If no response is detected, HC1 re-tries the same action. If HC1 receives an observation which is inconsistent or nonsensical, it re-tries the same action. Once HC1 fills both fields, it takes the corresponding *submit-x-y* action. A flow diagram of the logic used in HC1 is shown in Figure 4.16. HC2 is identical to HC1 except that if the machine receives an observation which is inconsistent or nonsensical, it immediately takes the fail action. HC3 employs a similar strategy to HC1 but extends HC1 by confirming each field as it is collected. If the user responds with “no” to a confirmation, it re-asks the field. If the user provides inconsistent information, it treats the new information as “correct” and confirms the new information. Once it has successfully filled and confirmed both fields, it takes the corresponding *submit-x-y* action.

Figure 4.17 shows the expected return for the handcrafted policies and the optimized POMDP solution vs. concept error rate (p_{err}). The optimized POMDP solution outperforms all of the handcrafted policies for all concept error rates. On inspection, conceptually the POMDP policy differs from the handcrafted policies in that it tracks conflicting evidence rather than discarding it. For example, whereas the POMDP policy can interpret the “best 2 of 3” observations for a given slot, the handcrafted policies can maintain only 1 hypothesis for each slot. As expected, the additional representational power of the POMDP is of no benefit in the presence of perfect recognition – note that when no recognition errors are made (i.e., $p_{err} = 0$), HC1 and HC2 perform identically to the POMDP policy. It is interesting to note that HC3, which confirms all inputs, performs least well for all concept error rates. For the reward function used in the test-bed system, requiring 2 consistent recognition results (the response to *ask* and the response to *confirm*) gives rise to longer dialogs which outweigh the benefit of the increase in accuracy.

4.5 Other POMDP-based dialog managers

In the literature, two past works have pioneered POMDP-based spoken dialog systems. First, Roy *et al* [92] cast the dialog manager of a robot in a nursing home environment as a POMDP. The POMDP’s 13 states represent a mixture of 6 user goals and various user actions, and its 20 actions include 10 performance-related actions (e.g., go to a different room, output information) and 10 clarifying questions. The POMDP’s 16 observations include 15 keywords and 1 non-sense word. The reward function returns -1 for each turn, +100 for the correct fulfilment of the user’s request, and -100 for incorrect fulfilment. Finding that an exact solution to this

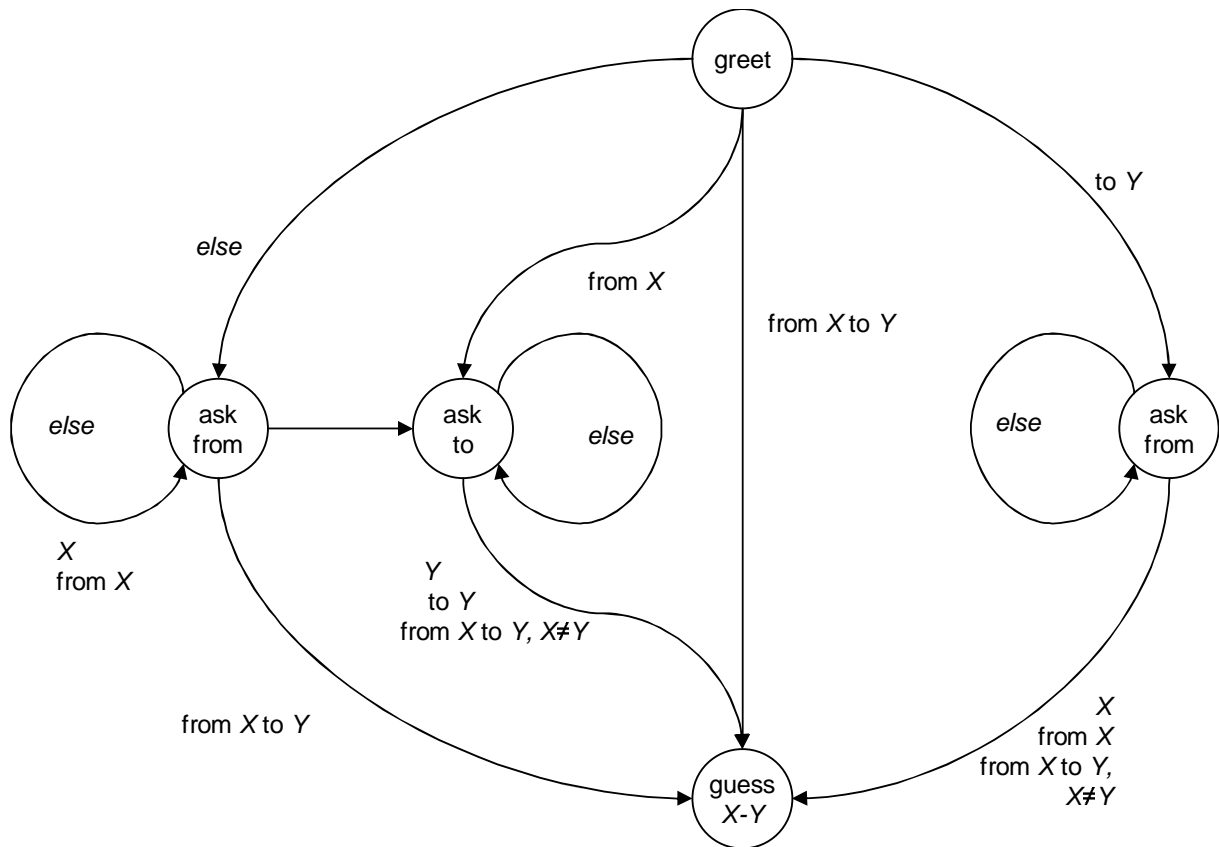


Figure 4.16 *HC1* handcrafted dialog manager baseline represented as a finite state controller. Node labels show the POMDP action to take for each node, and arcs show which POMDP observations cause which transitions. Note that the nodes in the diagram are entirely independent of the POMDP states.

POMDP is intractable, the authors perform optimization by casting the POMDP as an *Augmented MDP*, which performs planning by approximating belief space as a tuple $(s^*, H(b))$ where s^* is the state with the single highest probability $s^* = \arg \max_s b(s)$ and $H(b)$ is the entropy of the belief state $H(b) = -\sum_s b(s) \log_2 b(s)$. The entropy provides an indication of how accurate the hypothesis s^* , and since entropy is a real-valued quantity, the planning process partitions a single dimension rather than the entire belief simplex (of $|\mathcal{S}| = 13$ dimensions). The authors find that the reward gained per dialogue was significantly better for the Augmented MDP than for the MDP trained on the same data because the augmented MDP uses confirmation much more aggressively. Moreover, the authors find that the performance gain of the POMDP-based solution over the MDP-based solution is greater when speech recognition is worse, consistent with experiments above (Figure 4.6, page 49).

Second, Zhang *et al* [132, 133] create a POMDP-based dialog manager in a tour-guide spoken dialog system. The POMDP's 30 states are comprised of two components, 6 possible user goals and a "hidden" component taking one of 5 values, indicating the communicative status of the channel, such as *normal*, *error-silent*, or *error-noisy*. The POMDP's 18 actions include actions for asking the user's goal, confirming it, remaining silent, and submitting the user's goal. The

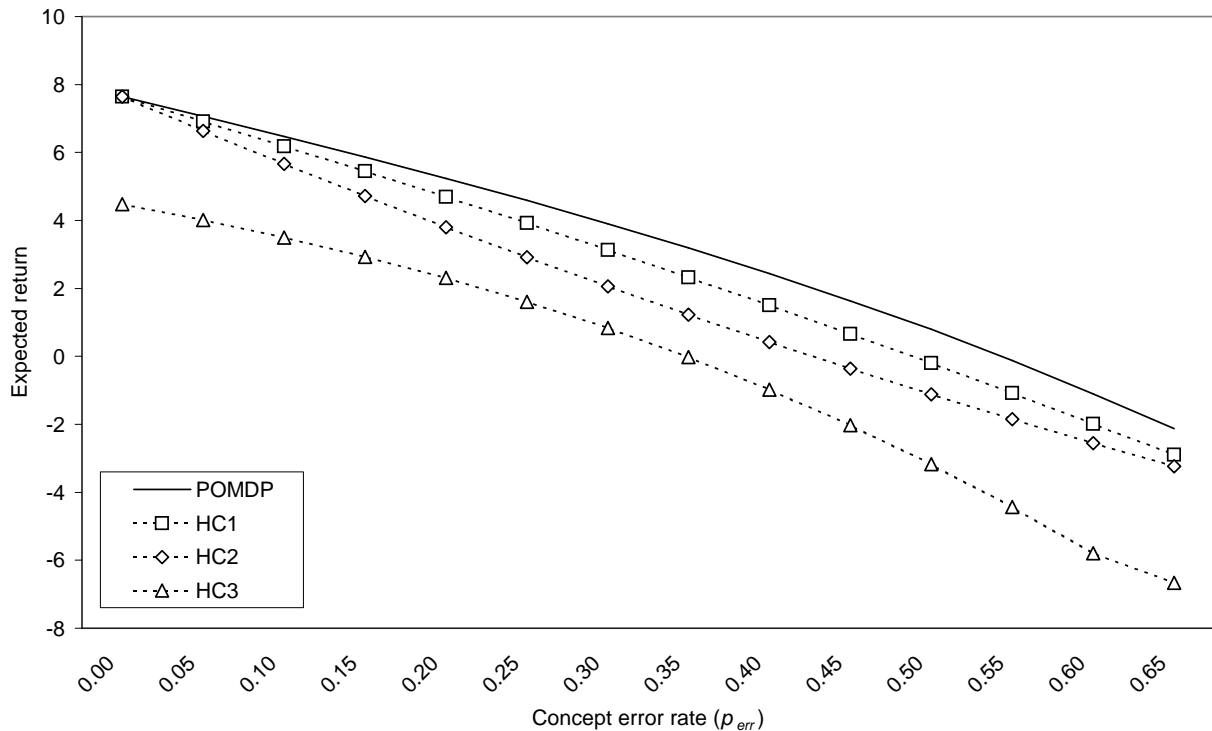


Figure 4.17 Concept error rate (p_{err}) vs. average return for POMDP and 3 handcrafted baselines.

POMDP’s 25 observations include statements of the user’s goal, the keywords *yes* and *no*, and indications of communicative trouble such as *no-signal*. Rewards are associated with each action, for example a small negative reward for querying the user’s goal, a large positive reward for correctly submitting the user’s goal and a large negative reward for incorrectly submitting the user’s goal. The authors find that exact optimizations are intractable, and optimize the POMDP using a grid-based approximation, which out-performs an MDP baseline.

The SDS-POMDP model is similar to these works in that the user’s goal is cast as an unobserved variable, speech recognition hypotheses are cast as POMDP observations, and system actions are cast as POMDP actions. However, the SDS-POMDP model extends these past works in several respects. First, the SDS-POMDP model factors s_m into three distinct components: the user’s goal s_u , the user’s action a_u , and the dialog history s_d . This factoring, combined with probabilistic decomposition, allows intuitive models of user action and the speech recognition process to be estimated from data (Equations 3.5, 3.6, 3.7, and 3.10, beginning on page 27). In the works above, an explicit component for user action does not appear, and as such it is unclear how interaction data could be used to estimate system dynamics. Next, the SDS-POMDP model includes a component for dialog history, s_d , absent from past work. By conditioning rewards on this term, the concept of “appropriateness” can be expressed. In other words, the SDS-POMDP model can make trade-offs between asking and confirming that accounts for both “appropriateness” and task completion, whereas past work accounts for only task completion when selecting actions. Finally, the SDS-POMDP model naturally accounts for continuous recognition features such as confidence score as continuous observations, whereas past models quantize confidence

score.

Finally, no past work has tackled the problem of scaling POMDPs to cope with spoken dialog systems of a real-world size, the subject of the remainder of the thesis.

4.6 Chapter summary

This chapter has compared the SDS-POMDP model to existing techniques for producing dialog managers. Like the SDS-POMDP model, many of these – parallel state hypotheses, confidence scoring, and automated planning – seek to address the uncertainty introduced by the speech recognition process by automating the action selection process (or a portion of it). However, theoretically each of these represents a special case or approximation to the SDS-POMDP model, and results from dialog simulation confirm that the increases in performance of the SDS-POMDP model are significant, especially as speech recognition errors become more prevalent. In addition, it has been shown how the SDS-POMDP model can be compared to handcrafted dialog managers in a principled way, and that the SDS-POMDP model indeed outperforms three handcrafted baselines. The SDS-POMDP model extends past applications of POMDPs to spoken dialog management, and the analyses in this chapter provide new support for POMDP-based spoken dialog management.

While the baseline TRAVEL application used in this chapter has reflected the important elements of a spoken dialog system, with only three cities it is too small to be of practical use. The next two chapters tackle the problem of scaling up the SDS-POMDP model to dialog problems of a useful size.

Scaling up: Summary point-based value iteration (SPBVI)

The previous chapter provided empirical support for the SDS-POMDP model using an example application which, although reflective of the key challenges faced by a real dialog system, was limited to 6 possible user goals – too small for real-world use. Adding more user goals renders straightforward optimization intractable, and indeed all past POMDP-based dialog management work has been limited to less than 10 user goals [92, 132, 133]. To scale to real-world problems, a more thoughtful optimization technique will be required.

This chapter addresses the problem of how to scale the SDS-POMDP model to a realistic size with a novel optimization technique called summary point-based value iteration (SPBVI). Rather than creating a plan over all of belief space as in typical POMDP optimization, SPBVI creates a plan in “summary space” which considers only the proportion of belief held by the single best hypothesis. The planner chooses actions in summary space and simple heuristics are used to map from summary space into (full) belief space. Because the size of summary space is small and fixed, plan complexity is tractable and independent of the number of possible user goals.

This chapter is organized as follows. First, consideration will be limited to a useful subclass of dialog problems called “slot-filling” dialogs, and it will be shown why POMDPs scale poorly within this class of dialogs. Next, it will be argued that the SDS domain has properties which can be exploited by an optimization algorithm, and then the details of the summary point-based value iteration (SPBVI) optimization algorithm will be presented. A larger version of the TRAVEL application called MAXITRAVEL is described and experiments from simulated dialogs then illustrate the ability of SPBVI to scale while retaining the advantages of a POMDP over baseline techniques.

5.1 Slot-filling dialogs

The SDS-POMDP model itself made no assumptions about the structure of the relationship between user goals, system actions, and user actions; however, in order to scale to larger problems some constraints will be needed. Consideration will be limited to so-called slot-filling dialogs.¹

¹Sometimes also called “form-filling” dialogs.

In a slot-filling dialog with W slots, \mathcal{S}_u can be restated as $\mathbb{S}_u, \mathbb{S}_u = \{(s_u^1, \dots, s_u^W)\}$ where $s_u^w \in \mathcal{S}_u^w$ and where \mathcal{S}_u^w refers to the set of values for slot w . The cardinality of slot w is given by $|\mathcal{S}_u^w|$. The user enters the dialog with a *goal*, a desired value for each slot, and the aim of the machine is to correctly determine the user's goal and *submit* it. Example slot-filling applications from the literature are shown in Table 5.1.

Domain	Example slots and values
Air travel (DARPA Communicator) [120]	FROM-CITY = {London, Boston, ...} TO-CITY = {London, Boston, ...} DIRECT = {true, false, ...} DATE = {..., 5 July 2006, ...} TIME = {..., 11-00, 11-30, ...} CLASS = {first, business, economy}
Computer buying [83]	FAMILY = {Mac, PC, ...} CPU-SPEED = {..., 1 GHz, 1.5 GHz, ...} PRICE = {\$599, \$649, ...} HDD-SPACE = {..., 100 GB, 120 GB, ...} MEMORY = {..., 512 MB, 1 GB, ...} TYPE = {desktop, laptop, tablet}
Room reservations (RoomLine) [8]	CAPACITY = {..., 10, 15, ...} START-TIME = {..., 11-00, 11-30, 12-00, ...} END-TIME = {..., 11-00, 11-30, ...} DATE = {..., 5 July 2006, ...} HAS-PROJECTOR = {true, false} BUILDING = {building-1, ...}

Table 5.1 Example slot-filling dialog domains.

As stated, the SDS-POMDP model scales poorly because its state space, action set, and observation set all grow as $\prod_w |\mathcal{S}_u^w|$. For example, in the TRAVEL application, the set of user goals \mathbb{S}_u represented all possible itineraries. Adding a new city to this set such as *oxford* requires new machine actions (such as $a_m = submit-from-cambridge-to-oxford$ and $a_m = confirm-to-oxford$) and new observations (such as $\tilde{a}_u = oxford$, $\tilde{a}_u = to-oxford$, and $\tilde{a}_u = from-london-to-oxford$).² In other words, the state space, action set, and observation set all grow as $O(|\mathbb{S}_u|) = O(\prod_w |\mathcal{S}_u^w|)$. In a world with 1000 cities, there are approximately 10^6 possible itineraries and thus on the order of 10^6 states, actions, and observations, making optimization completely hopeless, even with state of the art techniques. Worse, the additional state components reflecting the dialog history \mathcal{S}_d and user action \mathcal{A}_u exacerbate this growth.

A common technique in the POMDP literature to reduce the complexity of planning is to “compress” the state space by aggregating states [14, 37, 91, 87]. Unfortunately, in the dialog

²This example has assumed that someone would really want to go to Oxford.

domain, there is an important correspondence between states and actions, and this correspondence would be lost in a compression. For example a user goal such as $s_u = \text{from-london-to-edinburgh}$ has corresponding system actions such as *confirm-from-london* and *submit-from-london-to-edinburgh*, and it seems unlikely that an aggregated state such as *from-london-or-leeds-to-edinburgh* would be helpful. As a result, optimization techniques which attempt to compress the POMDP through state aggregation are bound to fail.

Looking through transcripts of simulated dialogs with the TRAVEL application, it was observed that actions like *confirm* and *submit* were only taken on the user goal with the highest belief mass. Intuitively, this is sensible: for confirmations, a “no” response increases belief state entropy, lengthening dialogs and decreasing return; and submitting a user’s goal incorrectly results in steep penalties. The intuition of the SPBVI method is to limit *a priori* actions like *confirm* and *submit* to act on only the most likely user goal. With this restriction, the proportion of belief mass held by the most likely goal is used for planning, but the actual value of the user goal is irrelevant. The structure of the slot-filling domain provides the framework required to map between actions and user goals.

Reducing this intuition to practice will be tackled in two distinct phases. First, in this chapter, dialogs with just one slot are considered, and the problem of scaling the *number of values* in a slot is addressed. The next chapter deals with how to scale the *number of slots*.

5.2 SPBVI method description

Summary point-based value iteration (SPBVI) consists of four phases: construction, sampling, optimization, and execution. In the construction phase, two POMDPs are constructed: a slot-filling SDS-POMDP called the master POMDP and a second, smaller POMDP called the summary POMDP, which compactly represents the belief in the most likely hypothesis. In the sampling phase, a random policy is used to sample belief points from the master POMDP. Each of these points is mapped into the summary POMDP, and in the optimization phase, optimization is performed directly on these points *in the summary POMDP*. Because the summary POMDP is much smaller, optimization remains tractable. In the execution phase, belief monitoring maintains a belief state in the master POMDP, this belief state is mapped into the summary POMDP, machine actions are selected in the Summary POMDP and then they are mapped back to the master POMDP. This section describes each of these phases in detail.

In the *construction* phase, a one-slot SDS-POMDP is created called the *master POMDP* with state space called *master space*. Here a few constraints will be added to the form of the SDS-POMDP components \mathcal{S}_u , \mathcal{A}_u , and \mathcal{A}_m . First, the set \mathcal{S}_u will consist of all possible slot values. Next the sets \mathcal{A}_u and \mathcal{A}_m will be defined as predicates which take elements of \mathcal{S}_u as arguments. For example, if \mathcal{S}_u is a set of airports, then the user action “I want London Heathrow” will be written $a_u = \text{state}(LHR)$, the user action “yes” will be written as $a_u = \text{yes}()$, and the user action “yes, Heathrow” will be written $a_u = \text{yes}(LHR)$. The set \mathcal{A}_m is similarly formed so that the machine action “Which airport do you want?” will be written $a_m = \text{ask}()$, the machine action

“Heathrow, is that right?” will be written $a_m = \text{confirm}(LHR)$, and the machine action which submits “Heathrow” will be written $a_m = \text{submit}(LHR)$. Finally, the set of dialog histories \mathcal{S}_d is not defined explicitly but it is assumed that the size of this set $|\mathcal{S}_d|$ is constant with respect to $|\mathcal{S}_u|$. For example, \mathcal{S}_d might track grounding as in the TRAVEL application.

Next, a parallel state space is formed, called *summary space*, consisting of 2 components: $\hat{\mathcal{S}}_u$ and $\hat{\mathcal{S}}_d$. The component $\hat{\mathcal{S}}_u$ consists of two elements, $\{\text{best}, \text{rest}\}$, and the component $\hat{\mathcal{S}}_d$ consists of the same elements as \mathcal{S}_d . A parallel action set $\hat{\mathcal{A}}_m$ is formed of the *predicates* of the members of \mathcal{A}_m . For example, if \mathcal{A}_m contains the actions $\text{confirm}(LHR)$, $\text{confirm}(LGW)$, etc., then $\hat{\mathcal{A}}_m$ contains the single corresponding action confirm .

Several functions map between master and summary space. First belief points in master space can be mapped to summary space with $bToSummary$ (Algorithm 4), which sets $\hat{b}(\hat{s}_u = \text{best})$ equal to the mass of the best hypothesis for the user’s goal, sets $\hat{b}(\hat{s}_u = \text{rest})$ equal to the mass of all of the other user goals, and sets $\hat{b}(s_d)$ to be simply a copy of $b(s_d)$. Actions can be mapped from master space to summary space by simply removing its argument, and from summary space to master space with $aToMaster$ (Algorithm 5), which adds the most likely user goal as an argument.

Algorithm 4: Function $bToSummary$.

Input: b
Output: \hat{b}
 // Sets $\hat{b}(\hat{s}_u = \text{best})$ to mass of most likely user goal in b .
 1 $\hat{b}(\hat{s}_u = \text{best}) \leftarrow \max_{s_u} b(s_u)$
 2 $\hat{b}(\hat{s}_u = \text{rest}) \leftarrow 1 - \hat{b}(\hat{s}_u = \text{best})$
 // Sets $\hat{b}(s_d)$ to $b(s_d)$.
 3 **foreach** $s_d \in \mathcal{S}_d$ **do**
 4 $\hat{b}(s_d) \leftarrow b(s_d)$

Algorithm 5: Function $aToMaster$.

Input: \hat{a}_m, b
Output: a_m
 1 **if** \hat{a}_m takes an argument in master space **then**
 2 $s_u^* \leftarrow \arg \max_{s_u} b(s_u)$
 3 $a_m \leftarrow \hat{a}_m(s_u^*)$
 4 **else**
 5 $a_m \leftarrow \hat{a}_m()$

The sampling phase will require another function, $sampleCorner$, which samples an unobserved state s and a belief state b given a state in summary space \hat{s} . In words, “sampleCorner” samples a state and a belief state which could map to a corner of summary space, \hat{s} .

Algorithm 6: Function *sampleCorner*.

```

Input:  $b, \hat{s}$ 
Output:  $b$ 
1  $s_u^* \leftarrow \arg \max_{s_u} b(s_u)$ 
2 if  $\hat{s}_u = \text{best}$  then
    // When  $\hat{s}_u = \text{best}$ , best guess for user's goal is correct:
    // set  $s_u$  to this guess and  $b(s_u)$  to that corner.
3   foreach  $s_u \in \mathcal{S}_u$  do
4      $b(s_u) \leftarrow 0$ 
5    $b(s_u^*) \leftarrow 1$ 
6 else
    // When  $\hat{s}_u = \text{rest}$ , best guess for user's goal is NOT correct:
    // set  $b(s_u^*)$  to zero and renormalize, then
    // sample  $s_u$  from this renormalized belief state.
7    $b(s_u^*) \leftarrow 0$ 
8    $\text{norm} \leftarrow \sum_{s_u} b(s_u)$ 
9   foreach  $s_u \in \mathcal{S}_u$  do
10     $b(s_u) \leftarrow \frac{b(s_u)}{\text{norm}}$ 
    // Copy  $\hat{b}(s_d)$  directly into  $b(s_d)$ .
11 foreach  $s_d \in \mathcal{S}_d$  do
12    $b(s_d) \leftarrow \hat{b}(s_d)$ 

```

In the *sampling* phase, similar to PBVI, a set of dialogs are run with a random policy to explore belief space and sample belief points which are likely to be encountered, but sampling in SPBVI differs in three respects: first, points are sampled in *summary* space; second, it is ensured that the *corners* of summary space are included in the sample; and third, *transition dynamics* are also sampled at each point. Sampling proceeds as in Algorithm 7, which first samples N distinct points in summary space by following a random policy. At each time-step, a point b in master space is sampled and mapped to \hat{b} in summary space using *bToSummary*. When a (sufficiently) new point in *summary* space is encountered it is recorded in \hat{b}_n , and system dynamics at that point are estimated using *samplePoint* (Algorithm 8). For POMDPs in general, estimating system dynamics – i.e., a distribution over successor belief states for a given action $P(b'|b, a)$ – requires enumerating every observation. This is infeasible for large SDS-POMDPs, and fortunately many observations will lead to the same successor belief state in summary space. For this reason, successor belief points are *sampled*: at each point \hat{b}_n , every action in $\hat{\mathcal{A}}_m$ is taken K times, and the resulting point in summary space $\hat{b}_n^{\hat{a},k}$ is recorded (ignoring the observation produced). The reward obtained is also recorded, in $\hat{r}_n^{\hat{a},k}$.³

³When a_m or \hat{a}_m is used in a superscript or subscript, it is shortened to a or \hat{a} , as in x_a or $x_{\hat{a}}$.

The motivation for including the corners in the samples arose from early experimentation with SPBVI. It was found that, when small numbers of belief points are sampled, regions of high certainty are sometimes omitted, resulting in policies which only take information gathering actions such as *ask* and never dialog-final actions such as *submit*. When this happens, SPBVI policies can produce dialogs which go on indefinitely; to prevent this, after N points have been sampled, it is then ensured that the *corners* of summary space have been visited, and any missing corners are sampled. Including the corners ensures that the policy will compute what actions to take when certainty is high, guaranteeing that the policy includes actions which complete the dialog. As more belief points are sampled (i.e., as N is increased), sampling the corners becomes redundant. Sampling of corners is done with the function *sampleCorner* (Algorithm 6), which finds a belief point b in master space which maps to a given corner of summary space \hat{s} .

Once sampling is complete, then for every element in $\{\hat{b}_n^{\hat{a},k}\}$, the index i of the closest \hat{b}_i is found, $\{l(n, \hat{a}_m, k)\}$. The result is a set $\{\hat{b}_n\}$ of belief points in summary space, a set $\{\hat{b}_n^{\hat{a},k}\}$ of successor belief points in summary space, a set $\{\hat{r}_n^{\hat{a},k}\}$ of rewards, and a set of indexes $\{l(n, \hat{a}_m, k)\}$ giving the closest point in $\{\hat{b}_n\}$ which approximates $SE(\hat{b}, \hat{a}_m, k)$ in summary space.

Optimization is performed in summary space as shown in Algorithm 9. A simplified, approximate form of value iteration is used which calculates expected return only at the belief points in $\{\hat{b}_n\}$. Like PBVI, in SPBVI a fixed-size set of N t -step conditional plans are iteratively generated for longer and longer time horizons (larger values of t). These back-ups are computed using the estimated dynamics of *summary* space $l(n, \hat{a}_m, k)$ and estimated reward function of summary space $\{\hat{r}_n^{\hat{a},k}\}$. However, unlike PBVI which computes the value of each conditional plan at all points in belief space, SPBVI estimates the value of a conditional plan *only at one point* in *summary space*. So rather than producing a value function consisting of a set of vectors $\{V_T^n(s)\}$ and corresponding actions $\{a_T^k\}$ like PBVI, SPBVI rather produces a set of *scalars* $\{\hat{v}_T^n\}$ where $\hat{v}_T^n \in \mathfrak{R}$ and actions $\{\hat{a}_T^k\}$ where $\hat{a}_T^k \in \hat{\mathcal{A}}_m$.⁴

To *execute* this policy, belief monitoring is performed in the master POMDP. To find the optimal action for a given belief point b , the corresponding summary belief point $\hat{b} = bToSummary(b)$ is computed, the closest summary belief point in the set $\{\hat{b}_n\}$ is found, its summary action \hat{a}^n is identified, and it is mapped to a master action $a_m = aToMaster(b, \hat{a})$. This process is detailed in Algorithm 10.

SPBVI is more efficient than PBVI because planning occurs in summary space, so complexity remains constant for any number of slot values $|S_u|$. Trade-offs between solution quality and computational complexity can be made via the number of belief points N and the number of observation samples K .

SPBVI can be viewed as a finite approximation to a so-called *belief MDP*. A belief MDP is a Markov decision process in belief (i.e., continuous) space with transition function $P(b'|b, a)$,

⁴Another version of SPBVI was explored which maintained *gradients* at each point like PBVI, but this version performed worse than maintaining scalar values. The cause of this was believed to be that the non-Markovian mapping to summary space caused gradients to be poorly estimated, and as a result the value of a conditional plan estimated for a given \hat{b} could be considerably over-estimated far from \hat{b} .

Algorithm 7: Belief point selection for SPBVI.

```

Input:  $\mathfrak{P}_{SDS}, \epsilon, N, K$ 
Output:  $\{\hat{b}_n\}, \{\hat{r}_n^{\hat{a},k}\}, \{l(n, \hat{a}_m, k)\}$ 
// First, sample a trajectory of points using a random policy.
1  $n \leftarrow 1$  // Initialize number of samples  $n$ .
2  $\hat{b}_n \leftarrow \text{bToSummary}(b_0)$  // Start with initial belief state  $b_0$ .
3  $(\{\hat{r}_n^{\hat{a},k}\}, \{\hat{b}_n^{\hat{a},k}\}) \leftarrow \text{samplePoint}(\mathfrak{P}_{SDS}, b_0, K)$ 
4  $s \leftarrow \text{sampleDist}_s(b)$ 
5  $b \leftarrow b_0$ 
6 while  $n < N$  do
    // Take a random action and compute new belief state.
7      $\hat{a}_m \leftarrow \text{randElem}(|\hat{\mathcal{A}}_m|)$ 
8      $a_m \leftarrow \text{aToMaster}(\hat{a}_m, b)$ 
9      $s' \leftarrow \text{sampleDist}_{s'}(P(s'|s, a_m))$ 
10     $o' \leftarrow \text{sampleDist}_{o'}(P(o'|s', a_m))$ 
11     $b \leftarrow SE(b, a_m, o')$ 
12     $\hat{b} \leftarrow \text{bToSummary}(b)$ 
    // If this is a (sufficiently) new point, add it to  $\hat{\mathcal{B}}$ .
13    if  $\min_{i \in [1, n]} |\hat{b}_i - \hat{b}| > \epsilon$  then
14         $n \leftarrow (n + 1)$ 
15         $\hat{b}_n \leftarrow \hat{b}$ 
16         $(\{\hat{r}_n^{\hat{a},k}\}, \{\hat{b}_n^{\hat{a},k}\}) \leftarrow \text{samplePoint}(\mathfrak{P}_{SDS}, b, K)$ 
17     $s \leftarrow s'$ 
// Second, sample corners  $\hat{s}$  of summary space
18 foreach  $\hat{s} \in \hat{\mathcal{S}}$  do
19     foreach  $\tilde{s} \in \hat{\mathcal{S}}$  do
20          $\hat{b}(\tilde{s}) \leftarrow 0$ 
21      $\hat{b}(\hat{s}) \leftarrow 1$ 
22     if  $\min_{i \in [1, n]} |\hat{b}_i - \hat{b}| > \epsilon$  then
23          $n \leftarrow (n + 1); N \leftarrow n$ 
24          $b \leftarrow \text{sampleCorner}(b_0, \hat{s})$ 
25          $\hat{b}_n \leftarrow \text{bToSummary}(b)$ 
26          $(\{\hat{r}_n^{\hat{a},k}\}, \{\hat{b}_n^{\hat{a},k}\}) \leftarrow \text{samplePoint}(\mathfrak{P}_{SDS}, b, K)$ 
// Finally, find index  $l(n, \hat{a}_m, k)$  of closest point to  $\hat{b}_n^{\hat{a},k}$ 
27 for  $n \leftarrow 1$  to  $N$  do
28     foreach  $\hat{a}_m \in \hat{\mathcal{A}}_m$  do
29         for  $k \leftarrow 1$  to  $K$  do
30              $l(n, \hat{a}_m, k) \leftarrow \arg \min_{\tilde{n}} |\hat{b}_n^{\hat{a},k} - \hat{b}_{\tilde{n}}|$ 

```

Algorithm 8: Function *samplePoint*.

```

Input:  $\mathfrak{P}_{SDS}, b, K$ 
Output:  $\{\hat{b}^{\hat{a},k}\}, \{\hat{r}^{\hat{a},k}\}$ 
// Iterate over all summary actions.
1 foreach  $\hat{a}_m \in \hat{\mathcal{A}}_m$  do
    // Take each summary action  $K$  times.
2   for  $k \leftarrow 1$  to  $K$  do
        // Sample a (possibly) new master state  $\tilde{s}$  from
        // the current belief state  $b$ .
3      $\tilde{s} \leftarrow \text{sampleDist}_{\tilde{s}}(b)$ 
        // Map summary action  $\hat{a}_m$  into master action  $a_m$ .
4      $a_m \leftarrow \text{aToMaster}(\hat{a}_m, b)$ 
        // Take action (sample new state and observation).
5      $s' \leftarrow \text{sampleDist}_{s'}(P(s'|\tilde{s}, a_m))$ 
6      $o' \leftarrow \text{sampleDist}_{o'}(P(o'|s', a_m))$ 
        // Compute the successor master state  $b'$ .
7      $b' \leftarrow SE(b, a_m, o')$ 
        // Save  $\hat{r}^{\hat{a},k}$  and  $\hat{b}^{\hat{a},k}$ 
8      $\hat{r}^{\hat{a},k} \leftarrow r(\tilde{s}, a_m)$ 
9      $\hat{b}^{\hat{a},k} \leftarrow \text{bToSummary}(b')$ 

```

defined as:

$$P(b'|b, a) = \sum_{o'} P(b'|b, a, o')P(o'|b, a) \quad (5.1)$$

$$= \begin{cases} P(o'|b, a) & \text{if } SE(b, a, o') = b' \\ 0 & \text{otherwise,} \end{cases} \quad (5.2)$$

and reward function on belief states $\rho(b, a)$, defined as:

$$\rho(b, a) = \sum_s b(s)r(s, a). \quad (5.3)$$

Producing an optimal infinite-horizon policy $\pi^*(b)$ for this belief MDP is equivalent to finding an optimal policy for the original POMDP on which it is based [2, 110, 51]. SPBVI estimates the continuous belief space in a belief MDP as a finite set of points $\{\hat{b}_n\}$, and estimates the belief

Algorithm 9: SPBVI optimization procedure.

Input: \mathfrak{P}_{SDS} , $\{\hat{b}_n\}$, $l(n, \hat{a}_m, k)$, $\{\hat{r}_n^{\hat{a},k}\}$, K , T

Output: $\{\hat{a}_t^n\}$

// Find the number of sampled points, N .

- 1 $N \leftarrow |\{\hat{b}_n\}|$
- // Set initial value estimate for each point to 0.
- 2 **for** $n \leftarrow 1$ **to** N **do**
- 3 $\hat{v}_0^n \leftarrow 0$
- // Iterate over horizons from 1 to T .
- 4 **for** $t \leftarrow 1$ **to** T **do**
- // Generate $\{\hat{v}^{\hat{a},n}\}$, values of all possibly useful CPs.
- 5 **for** $n \leftarrow 1$ **to** N **do**
- 6 **foreach** $\hat{a}_m \in \hat{\mathcal{A}}_m$ **do**
- // Value of taking action \hat{a}_m from point n with
- // t time-steps to go is average of sampled immediate
- // rewards $\hat{r}_n^{\hat{a},k}$ and discounted $t-1$ step return $\hat{v}_{t-1}^{l(n,\hat{a},k)}$.
- 7 $\hat{v}^{\hat{a},n} \leftarrow \left(\frac{1}{K} \sum_k \hat{r}_n^{\hat{a},k} \right) + \left(\frac{\gamma}{K} \sum_k \hat{v}_{t-1}^{l(n,\hat{a},k)} \right)$
- // Prune $\{\hat{v}^{\hat{a},n}\}$ to yield $\{\hat{v}_t^n\}$, values of actually useful CPs.
- 8 **for** $n \leftarrow 1$ **to** N **do**
- 9 $\hat{a}^* \leftarrow \arg \max_{\hat{a}} \hat{v}^{\hat{a},n}$
- 10 $\hat{a}_t^n \leftarrow \hat{a}^*$
- 11 $\hat{v}_t^n \leftarrow \hat{v}^{\hat{a}^*,n}$

Algorithm 10: SPBVI action selection procedure, used at runtime.

Input: b , $\{\hat{b}_n\}$, $\{\hat{a}_n\}$

Output: a_m

// Map current (master) belief state b to

// summary belief state \hat{b} .

- 1 $\hat{b} \leftarrow \text{bToSummary}(b)$
- // Find index n^* of the closest sampled point \hat{b}_n .
- 2 $n^* \leftarrow \arg \min_n |\hat{b}_n - \hat{b}|$
- // The best summary action is \hat{a}^{n^*} ; map this
- // summary action to its corresponding master action a_m .
- 3 $a_m \leftarrow \text{aToMaster}(b, \hat{a}^{n^*})$

MDP's transition function and reward function on those points as $P(\hat{b}_i|\hat{b}_n, \hat{a})$ and $\rho(\hat{b}_n, \hat{a})$:

$$P(\hat{b}_i|\hat{b}_n, \hat{a}) \approx \frac{1}{K} \sum_k eq(l(n, \hat{a}, k), i), \quad (5.4)$$

$$\text{where } eq(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

$$\rho(\hat{b}_n, \hat{a}) \approx \frac{1}{K} \sum_k \hat{r}_n^{\hat{a}, k} \quad (5.5)$$

SPBVI differs from an Augmented MDP [92] in two important respects. First, an Augmented MDP summarizes the belief state using entropy, whereas SPBVI summarizes the belief state as the proportion of belief mass held by the most likely state. Since reward depends on the likelihood that the best guess is correct, the distribution of belief among the less likely states (which entropy accounts for) seems unimportant. More crucially, an Augmented MDP does not compress actions or observations, and as such is challenged to scale to dialog problems of a realistic size.

Given this formulation, the SPBVI optimization procedure itself might be properly described as *MDP* optimization; however SPBVI distinguishes itself from a typical MDP in that its states are (approximations to) points in belief space using a proper state estimator. Even so, SPBVI faces three potential limitations. First, like PBVI, SPBVI optimizes actions for a finite set of points $\{\hat{b}_n\}$ and not the entire belief simplex. As such it is always possible that a conditional plan which is optimal for a region which doesn't include a belief point will be omitted. Second, as described above, unlike PBVI, SPBVI computes only the value of a conditional plan at each point, and not its value gradient. As a result, SPBVI does not compute accurate boundaries between regions. In other words, action selection for new belief points must involve a heuristic (such as the "nearest neighbor" approach described here), and this heuristic introduces errors into the dynamic programming procedure. Finally, since the summary belief state is a non-linear function of master belief state, the dynamics of summary space are not guaranteed to be Markovian. As a result, the central Markov assumption of value iteration may be violated and value iteration may fail to produce good policies. For these reasons, it is important to test the algorithm empirically to understand whether these potential theoretical limitations are problematic in practice.

5.3 Example SPBVI application: MAXITRAVEL

To evaluate SPBVI a one-slot SDS-POMDP called MAXITRAVEL was created in which \mathcal{S}_u consists of 100 airports, and the machine's task is to correctly identify which airport the user wants information about. As in the previous chapter, the user's goal is chosen randomly from a uniform distribution at the beginning of the dialog, and is constant throughout the dialog.

The user action set \mathcal{A}_u consists of predicates which take elements of \mathcal{S}_u as arguments and these are listed in Table 5.2. As in the TRAVEL application, the dialog history \mathcal{S}_d includes $\{n, u, c\}$ where n means not stated, u means unconfirmed, and c means confirmed. The machine action set \mathcal{A}_m includes three predicates, listed in Table 5.3.

\hat{a}_u	Example a_u	Utterance form of example a_u
<i>state</i>	<i>state(LHR)</i>	“London Heathrow”
<i>yes</i>	<i>yes()</i>	“Yes”
<i>yesState</i>	<i>yesState(LHR)</i>	“Yes, London Heathrow”
<i>no</i>	<i>no()</i>	“No”
<i>noState</i>	<i>noState(LHR)</i>	“No, London Heathrow”
<i>null</i>	<i>null()</i>	(User says nothing)

Table 5.2 *User actions in the MAXITRAVEL application.*

\hat{a}_m	Example a_m	Utterance form of example a_m
<i>ask</i>	<i>ask()</i>	“Which airport do you want?”
<i>confirm</i>	<i>confirm(LHR)</i>	“London Heathrow, is that right?”
<i>submit</i>	<i>submit(LHR)</i>	(Machine gives info about Heathrow and ends dialog)

Table 5.3 *Machine actions in the MAXITRAVEL application.*

The user action model used in MAXITRAVEL $P(a'_u|s'_u, s_d, a_m)$ was estimated from real dialog data collected in the SACTI-1 corpus [125]. The SACTI-1 corpus contains 144 dialogs in the travel/tourist information domain using a “simulated ASR channel” which introduces errors similar to those made by a speech recognizer [112]. One of the subjects acts as a tourist seeking information (analogous to a user) and the other acts as an information service (analogous to a spoken dialog system), and the behaviors observed of the subjects in the corpus are broadly consistent with behaviors observed of a user a real spoken dialog system [125]. Wizard/User turn pairs which broadly matched the types of action in \mathcal{A}_m and \mathcal{A}_u were annotated. The annotations were divided into a “training” set which is used in this chapter and a “testing” set which will be used at the end of the next chapter as held-out data for an evaluation. From the training data a user model $P(a'_u|s'_u, a_m)$ was estimated using frequency counting (i.e., maximum likelihood), shown in Table 5.4. Due to data sparsity in the SACTI-1 corpus, the user acts *yes* and *no* were grouped into one class, so probabilities for these actions are equal (with appropriate conditioning for the sense of *yes* vs. *no*).

The reward function provided a large positive reward (+12.5) for taking a correct submit action; a large penalty (−12.5) for taking an incorrect submit action; and a host of smaller penalties depending on the appropriateness of information gathering actions. The complete reward function is summarized in Table 5.5.

The ASR model introduced in the TRAVEL application was used again in MAXITRAVEL (Equation 3.15, page 33). Recall this model is parameterized by concept error rate (p_{err}) which gives the probability of making a concept error and h which sets the informativeness of the confidence score. A discount of $\gamma = 0.99$ was selected.

SPBVI takes three optimization parameters: N , the number of summary points to sample; K

Machine Action		User Response		$P(a_u' s_u' = LHR, a_m)$
Utterance	a_m	Utterance	a_u'	
“Which airport?”	<code>ask()</code>	“Heathrow” (User says nothing)	<code>LHR</code>	0.987
		“Yes” (User says nothing)	<code>yes()</code>	0.013
“Heathrow, is that right?”	<code>confirm(LHR)</code>	“Yes, Heathrow” (User says nothing)	<code>yesState(LHR)</code>	0.782
		“No” (User says nothing)	<code>null</code>	0.205
“Gatwick, is that right?”	<code>confirm(LGW)</code>	“No, Heathrow” (User says nothing)	<code>no()</code>	0.013
		“Yes, Heathrow” (User says nothing)	<code>noState(LHR)</code>	0.782
				0.205
				0.013

Table 5.4 Summary of user model parameters for the MAXITRAVEL application. The user goal LHR corresponds to London Heathrow airport and LGW corresponds to London Gatwick airport.

Machine action (a_m)	Dialog History (s_d)	Description	$r(s_u = \mathbf{x}, s_d, a_m)$
$ask()$	n	Mach. asks for value not yet stated	-1
	u	Mach. asks for value stated but not confirmed	-2
	c	Mach. asks for value already confirmed	-3
$confirm(y)$	n	Mach. confirms a value not yet stated	-3
	u	Mach. confirms value stated but not confirmed	-1
	c	Mach. confirms value already confirmed	-2
$submit(y),$ $y = x$	—	Mach. submits correct value	+12.5
$submit(y),$ $y \neq x$	—	Mach. submits incorrect value	-12.5

Table 5.5 *Reward function for the MAXITRAVEL application. Mach. stands for “Machine”; the dash (—) indicates that the row corresponds to any value.*

the number of observation samples at each point \hat{b}_n ; and T , the length of the planning horizon.⁵ In the previous chapter it was found empirically for PBVI that $T = 50$ produced asymptotic performance on a very similar problem. The notion of planning horizon in PBVI and SPBVI is analogous and $T = 50$ was used again in this chapter. However the number of belief points N is *not* analogous because PBVI constructs *vectors* at each point whereas SPBVI only estimates a single value, so appropriate values of N and K for MAXITRAVEL need to be determined.

First, the confidence score informativeness was set to $h = 0$, the number of observation samples was set to $K = 50$, and N was varied. Results are shown in Figure 5.1, which shows that $N = 100$ achieved asymptotic performance for all concept error rates (p_{err}). Then the concept error rate was set to $p_{err} = 0.30$, and the experiment was repeated for various values of confidence score informativeness h . Results are shown in Figure 5.2, and again indicate that $N = 100$ seems to provide asymptotic performance.

Next, the number of belief points was set to $N = 100$ and optimization was run for various numbers of observation samples K . Results are shown in Figure 5.3. Most of the variations for $K \geq 20$ are within bounds of error estimation, and overall $K = 50$ appears to achieve asymptotic performance for all concept error rates (p_{err}). This was repeated for various values of confidence score informativeness h with concept error rate $p_{err} = 0.30$. Results are shown in Figure 5.4, and again indicate that $K = 50$ appears to provide asymptotic performance for a range of operating conditions.

Based on these findings, the values $K = 50$ and $N = 100$ were used for the rest of the experiments in this chapter. Figures 5.5 and 5.6 show sample conversations between a user and a dialog manager created with SPBVI for the MAXITRAVEL application using these values, for a

⁵The parameter ϵ was set as in PBVI: $\epsilon = \frac{1}{50 \cdot N}$; experimentation showed that any reasonably small value of ϵ performed similarly.

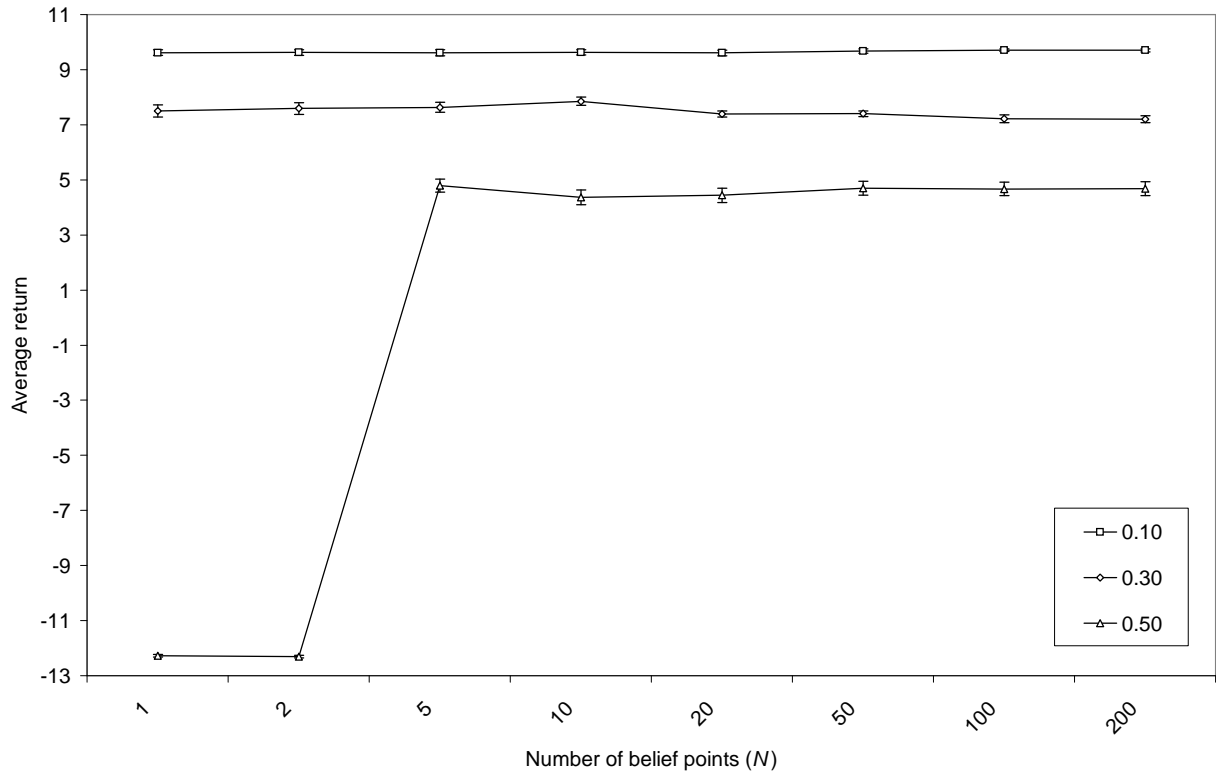


Figure 5.1 Number of belief points N vs. average return for various concept error rates (p_{err}) for the MAXITRAVEL application optimized with SPBVI with $K = 50$ observation samples.

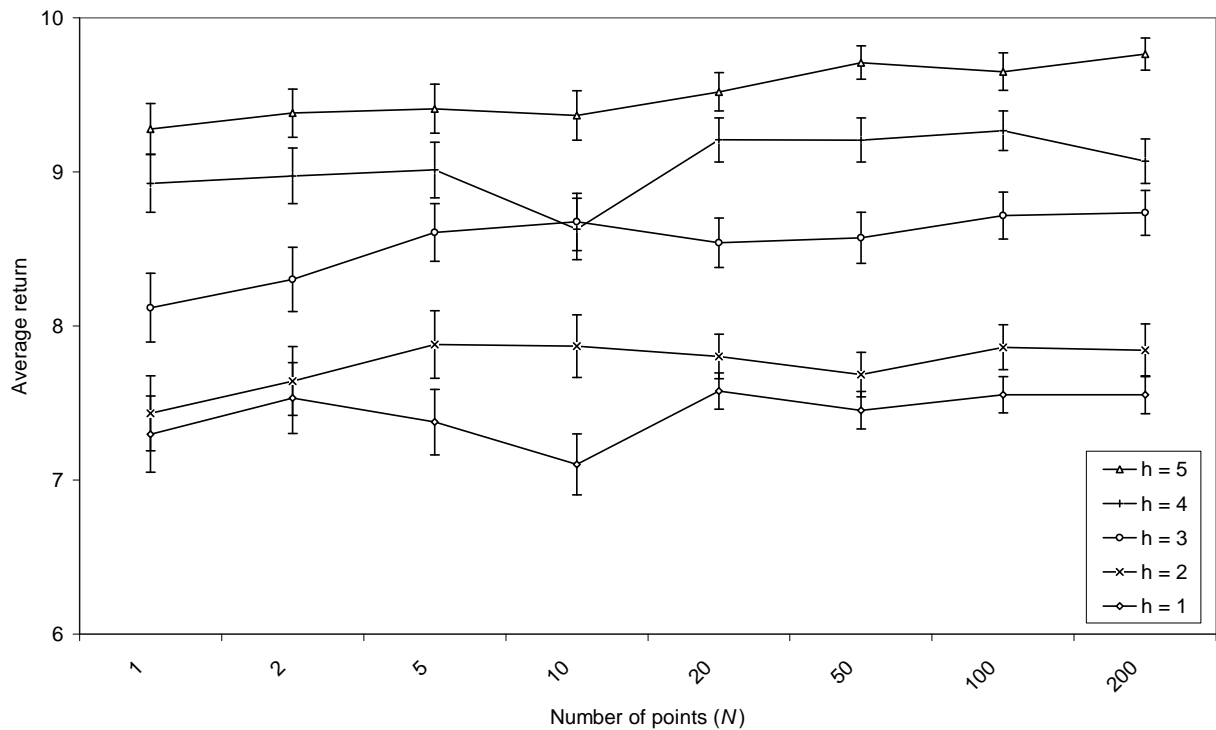


Figure 5.2 Number of belief points N vs. average return for various values of h for the MAXITRAVEL application optimized with SPBVI with $K = 50$ observation samples (Concept error rate $p_{err} = 0.30$).

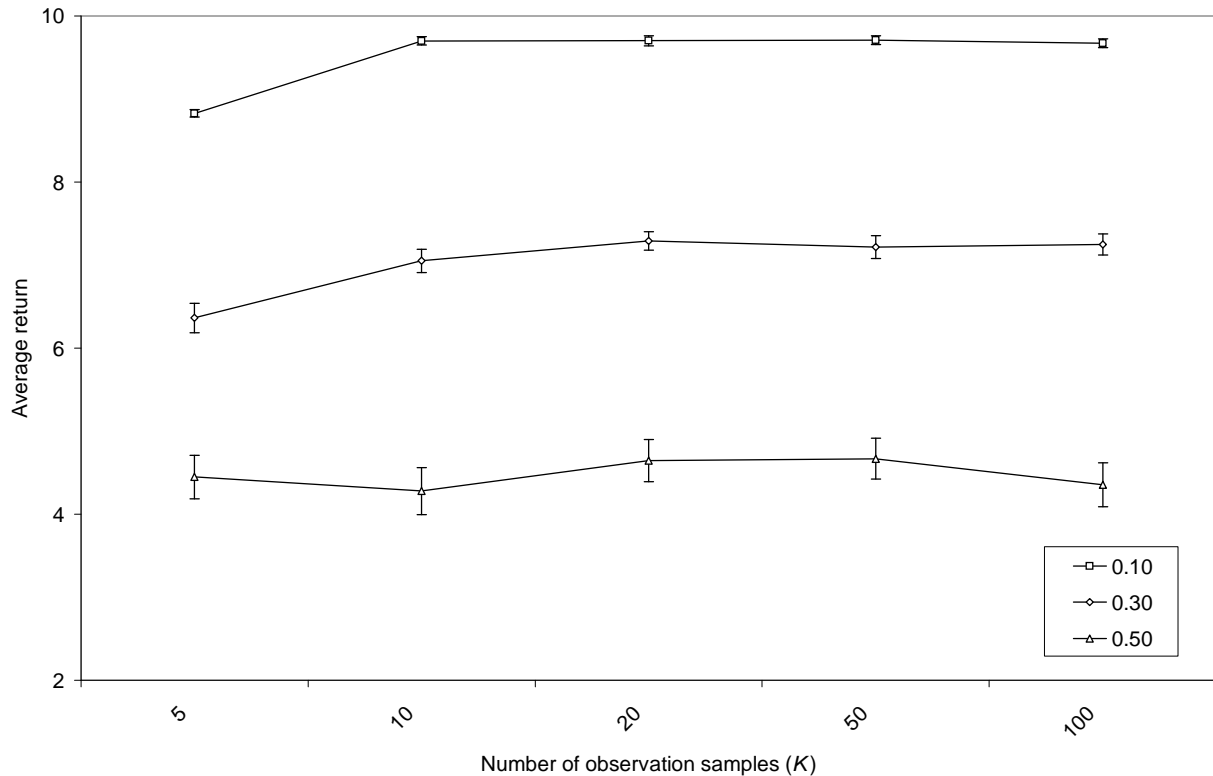


Figure 5.3 Number of observation samples K vs. average return for various concept error rates (values of p_{err}) for the MAXITRAVEL application optimized with SPBVI with $N = 100$ belief points.

reasonable concept error rate and somewhat informative confidence score ($p_{err} = 0.30, h = 2$). There are 100 user goals (i.e., airports), of which three are shown in Figures 5.5 and 5.6. Initially belief mass is spread evenly over all these user goals, and the belief mass for the *best* component in summary space equals the mass of (any) one of these. At each time-step, the policy takes the summary belief state as input and for the initial belief state the policy selects the *ask* action.

In the first dialog (Figure 5.5), after the “ask” action in M1, the first recognition in U1 of “Heathrow” is accurate and recognized with a reasonably high confidence score (0.87). Belief updating is performed in master space and belief mass shifts toward the value *LHR* (which stands for “London Heathrow airport”). After the belief update, *LHR* is the user goal with the highest belief mass, and its belief mass accounts for the *best* value in summary space ($\hat{b}(\hat{s}_u = \text{best})$), shown on the right side of Figure 5.5. Based on the summary belief state, the policy selects the *confirm* action, which is mapped to the *confirm(LHR)* action (M2) because *LHR* is the user goal with the highest belief mass. The user’s response of “yes” (U2) is correctly recognized and the machine then takes the *submit(LHR)* action and completes the dialog successfully.

The second dialog (Figure 5.6) shows an example of a recognition error. The user’s response to the initial *ask* action (M1), “Boston” (U1), is mis-recognized as *LHR* with a moderate confidence score (0.51). As before, this causes belief mass to shift toward *LHR* in master space but the shift is somewhat less than in the first dialog (Figure 5.5) since the confidence score here is lower. The resulting belief state in master space is mapped to summary space, and the summary

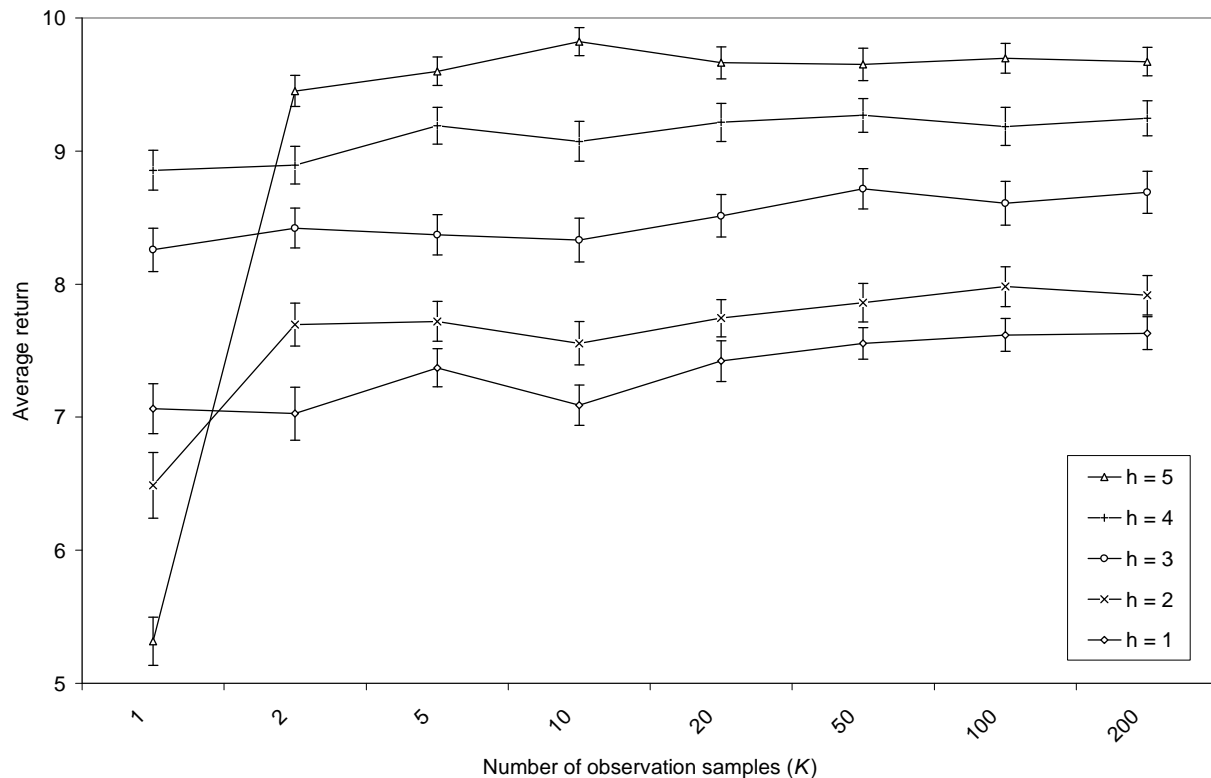


Figure 5.4 Number of observation samples K vs. average return for various levels of confidence score informativeness (h) for the MAXITRAVEL application optimized with SPBVI with $N = 100$ belief points (Concept error rate $p_{err} = 0.30$).

belief state is passed to the policy for action selection. For this summary state, the policy again chooses to *confirm* (M2). The user’s response to this confirmation, “No, Boston” (U2) is correctly recognized with moderate confidence score (0.53). This results in a shift of belief mass away from *LHR*; some of this shift is toward *BOS*, but since the user model indicates that responses like “No, Boston” (vs. just “No”) are relatively unlikely, the increase in belief mass for *BOS* is moderate, and much of the belief mass taken from *LHR* is spread out over all other user goals. Even so, *BOS* is now the user goal with the highest belief mass and its mass is mapped to the *best* value in summary space, where the policy selects the action *ask* (M3). Intuitively the *ask* action seems reasonable here since belief in any one user goal is rather low, and lower than in U2. The user’s response “Boston” (U3), which is understood correctly, causes mass to shift toward *BOS*, and when mapped into summary space the policy chooses the *submit* action.

5.4 Comparisons with baselines

To assess the performance of SPBVI quantitatively, a similar set of comparisons will be made using the MAXITRAVEL application as were made using the TRAVEL application in the previous chapter. First, a comparison with an MDP *excluding* confidence score quantifies the benefit of maintaining multiple hypotheses. Second, a comparison with an MDP *including* confidence

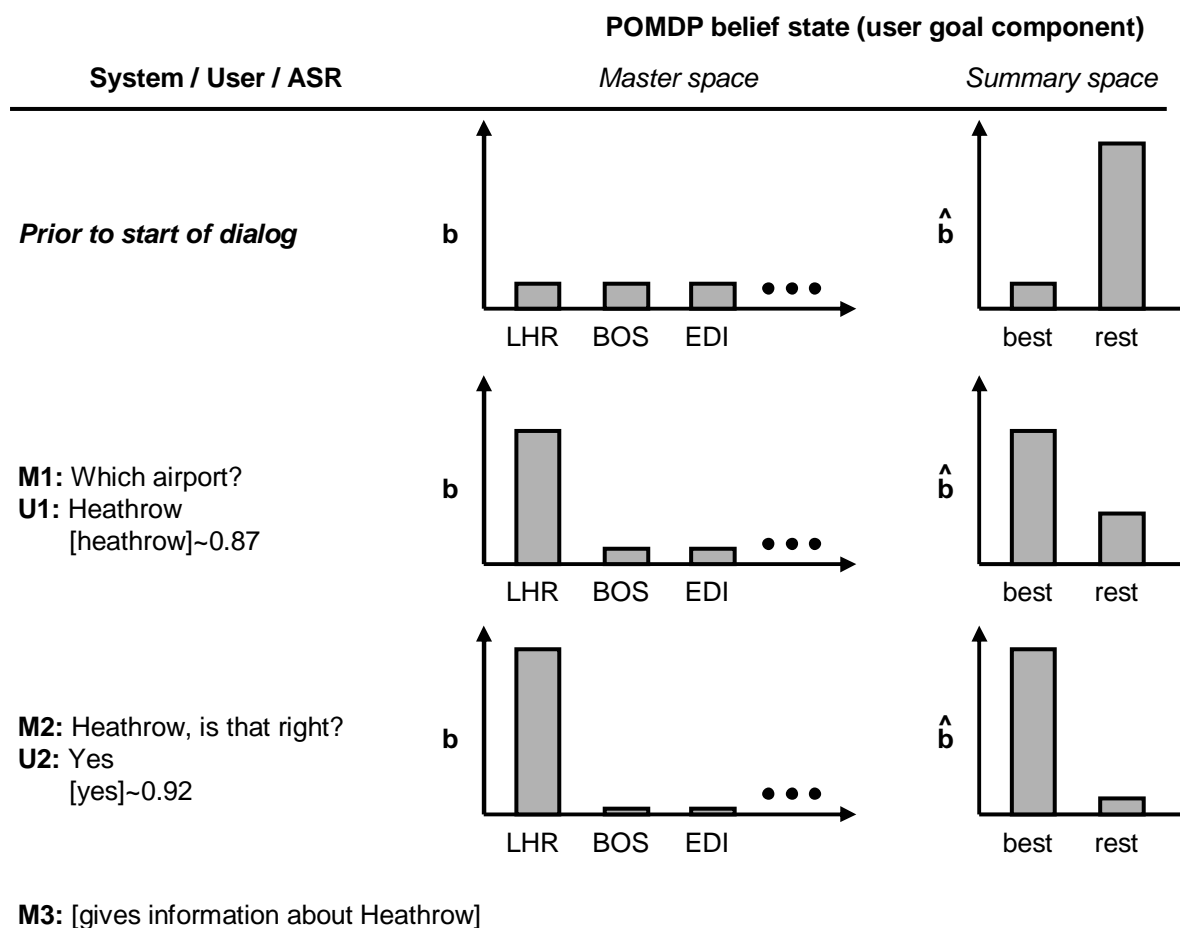


Figure 5.5 Sample conversation between user and SPBVI-based dialog manager (1 of 2). LHR stands for “Heathrow”, BOS stands for “Boston”, and EDI stands for “Edinburgh”.

score tests how well confidence score information is exploited. Finally, a comparison with two handcrafted controllers assess the overall benefit of automated planning.

SPBVI was first compared to an MDP with no contribution of confidence score information (i.e., with $h = 0$). The MDP baseline was formed as explained in section 4.3 (page 54) simplified for a single slot, and as before optimization was performed by running 50,000 simulated dialogs using Watkins’ Q-learning [121]. The resulting SPBVI and MDP policies were then each evaluated by simulating 10,000 dialogs. This process was repeated for concept error rates (p_{err}) ranging from 0.00 to 0.65, and results are shown in Figure 5.7. The Y-axis shows average reward gained per dialog, and error bars indicate the 95% confidence interval for the true average reward gained per dialog turn. As in the TRAVEL application, when no recognition errors are made (i.e., $p_{err} = 0$), the MDP and POMDP perform equivalently, demonstrating that in the presence of perfect recognition, there is no benefit in maintaining multiple hypotheses. However as concept recognition errors become more common (i.e., as p_{err} increases), the SPBVI policy outperforms the MDP policy by an increasing margin. This trend indicates that the heuristic introduced in the SPBVI method (i.e., limiting machine actions to operate on only the best hypothesis) enables policy optimization to scale without forfeiting the POMDP’s ability to effectively exploit

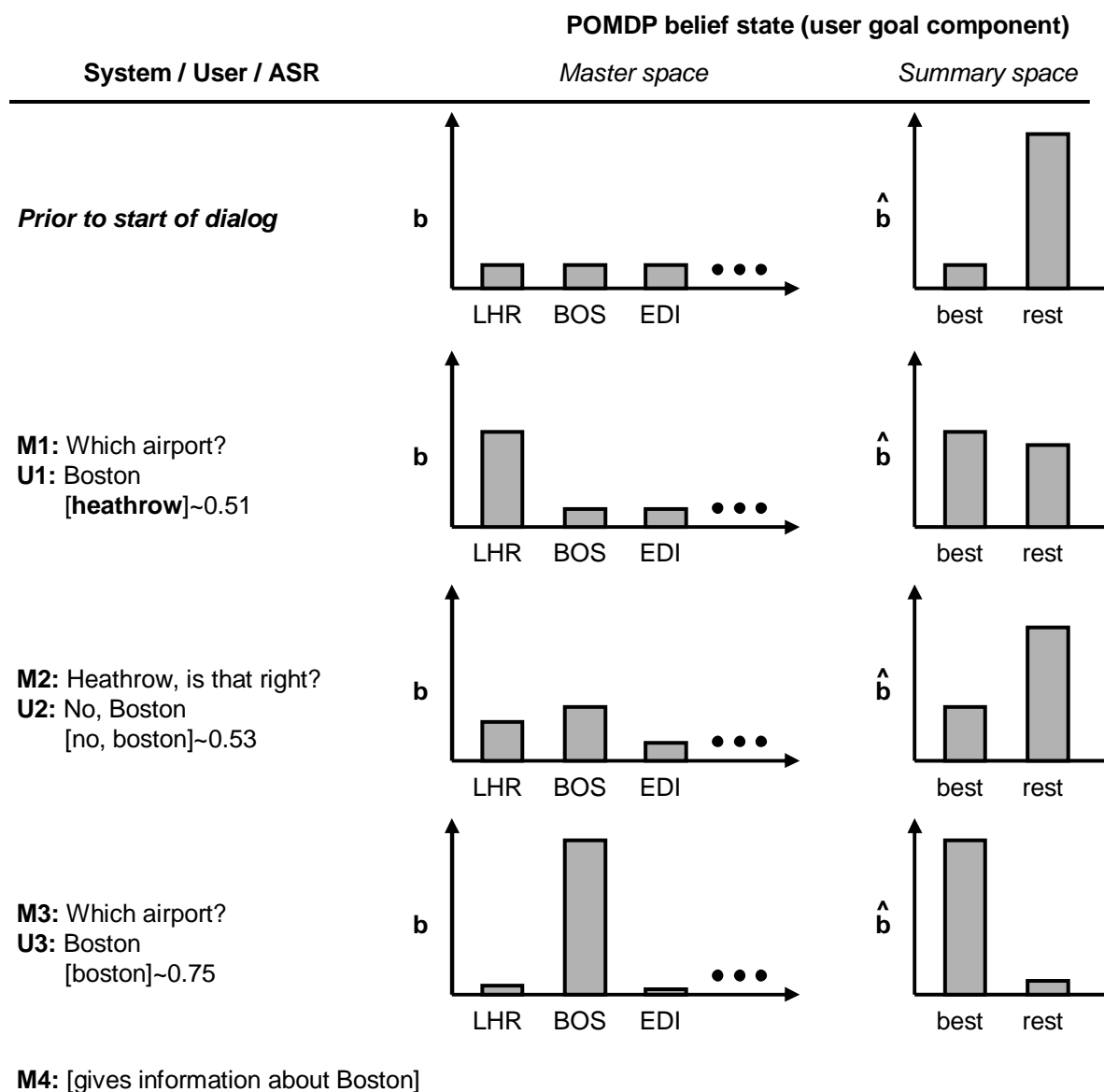


Figure 5.6 Sample conversation between user and SPBVI-based dialog manager (2 of 2). LHR stands for “Heathrow”, BOS stands for “Boston”, and EDI stands for “Edinburgh”.

uncertainty in the dialog state when selecting actions.

Next, confidence score information was added. The “MDP-2” baseline was used as explained in section 4.2 (page 48), again simplified for a single slot, and trained as in the previous experiment. Figure 5.8 shows informativeness of the confidence score h vs. the average return of the SPBVI and MDP-2 policies for various concept error rates. When concept recognition errors are present (i.e., $p_{err} > 0$), the POMDP outperforms the MDP-2 baseline, and as the confidence score becomes more informative (i.e., as h increases), performance at a given concept error rate (p_{err}) increases for both the POMDP and MDP-2 policies, with greater increases observed for higher error rates. However, when recognition is perfect (i.e., when $p_{err} = 0$), the performance of the MDP-2 and POMDP is constant with respect to h , illustrating that when no recognition

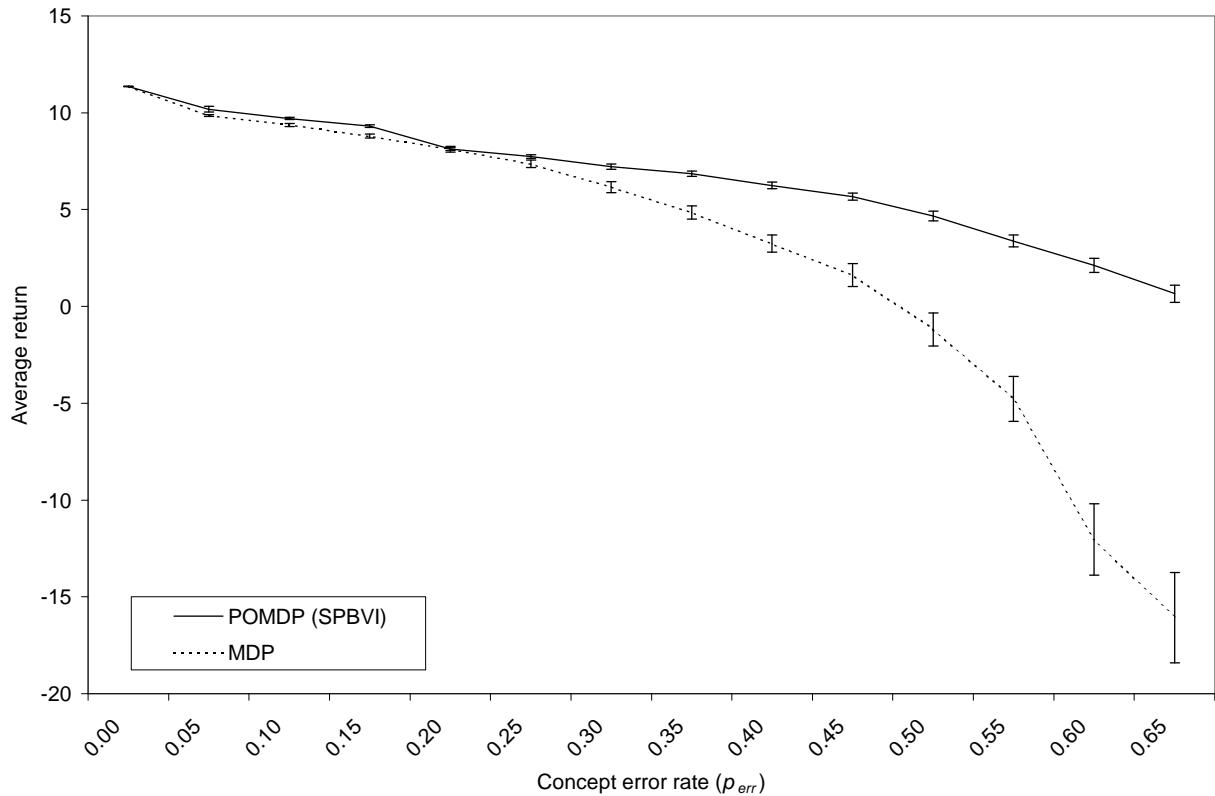


Figure 5.7 Concept error rate (p_{err}) vs. average return for the SPBVI and MDP methods with no confidence score.

errors are made, confidence score adds no further information.

To assess the overall benefit of automated planning, SPBVI optimization was then compared to two handcrafted dialog managers. Confidence score information was not used (i.e., h was set to 0). Two hand-crafted baseline controllers were created, HC4 and HC5. HC4 (Figure 5.9) first takes the ask action, and if an airport x is recognized, the action $confirm(x)$ is taken. If the observation $yes()$ or $yesState(x)$ is then recognized, the action $submit(x)$ is taken, ending the dialog. If any other observation is received, the process starts from the beginning. HC5 (Figure 5.10) begins the same, but after recognizing an airport x , it instead takes the $ask()$ action again and if a consistent airport is recognized, then the action $submit(x)$ is taken. If any other observation is received, HC5 continues to take the ask action until it receives two consistent sequential responses.

For the small TRAVEL application it was possible to assess the handcrafted controllers analytically (via Equation 4.3 on page 59), but here the set of linear equations is too large to solve so HC4 and HC5 were instead evaluated empirically by running 10,000 simulated dialogs. Results for concept error rates (p_{err}) ranging from 0.00 to 0.65 are shown in Figure 5.11. The Y-axis shows average reward gained per dialog, and error bars indicate the 95% confidence interval. The POMDP solution outperforms both handcrafted controllers at all error rates. It is interesting to note that at lower error rates, HC4 outperforms HC5 whereas at higher error rates the opposite is true. In other words, at lower error rates it is better to confirm recognition hypotheses,

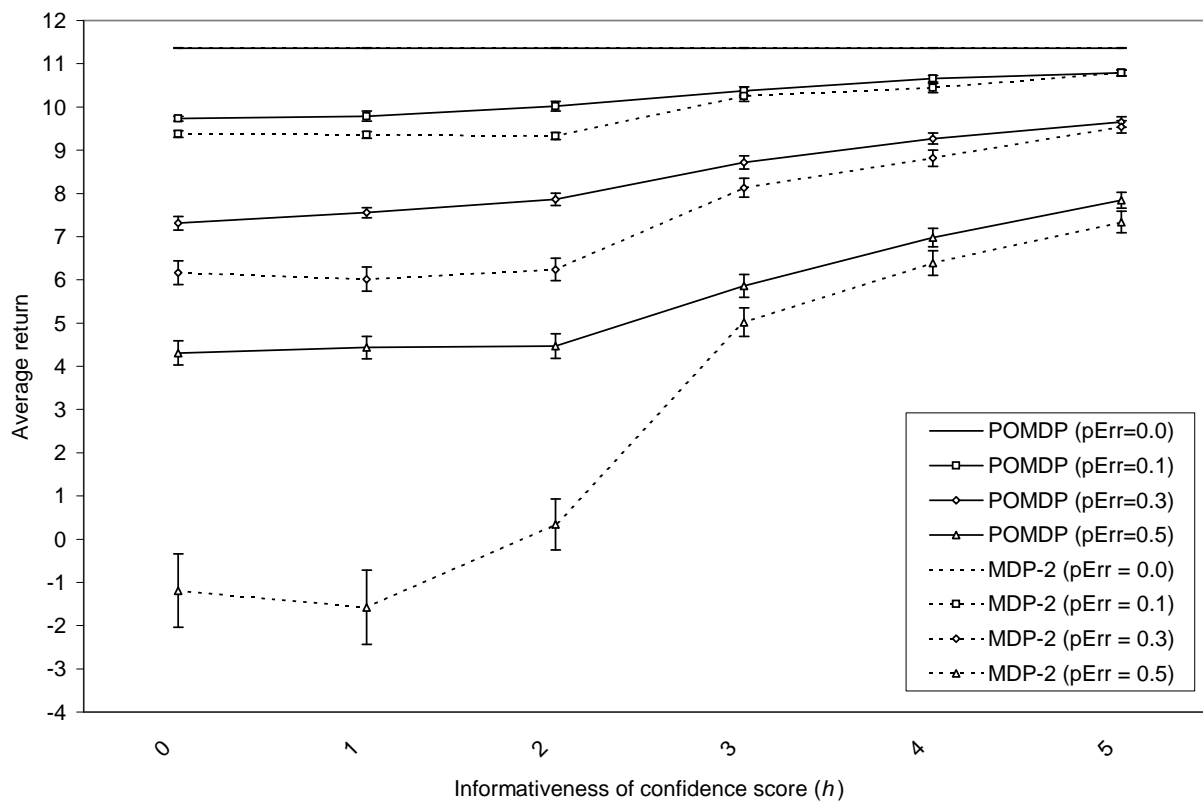


Figure 5.8 Confidence score informativeness h vs. average return for SPBVI and MDP-2 baselines for various concept error rates (p_{err}).

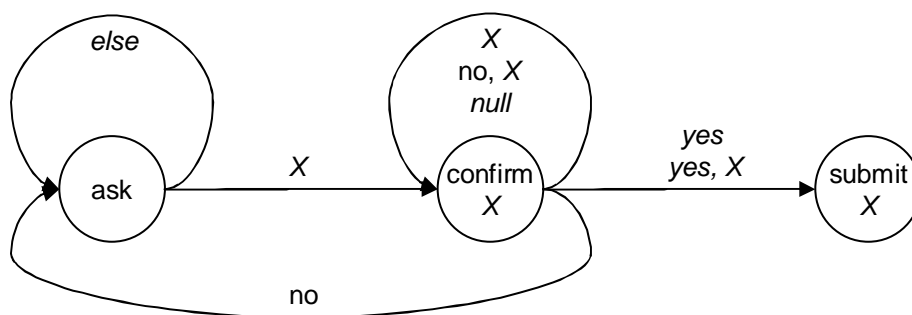


Figure 5.9 The HC4 handcrafted dialog manager baseline. Control starts in the left-most node.

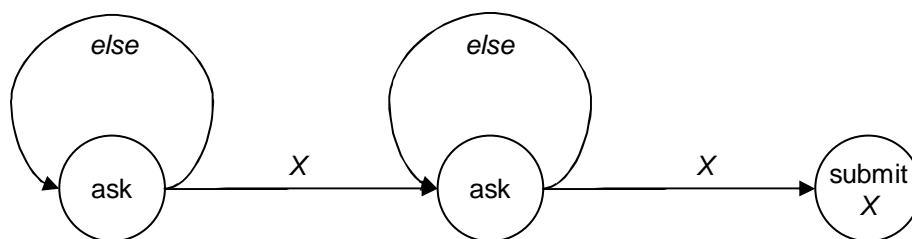


Figure 5.10 The HC5 handcrafted dialog manager baseline. Control starts in the left-most node.

whereas at higher error rates it is better to repeatedly take the $ask()$ action until two consistent, consecutive recognition hypotheses are observed. This is a consequence of the per-turn penalties used in the reward function, and the amount of information gained by each of these two actions. Taking a second $ask()$ action incurs a higher per-turn penalty than taking the $confirm(x)$ action (-3 vs. -1). Yet when a mis-recognition occurs, a user response of $no()$ to a $confirm()$ action provides no new hypothesis for a user goal, whereas responses to an $ask()$ action *can* provide an alternate (possibly valid) hypothesis. This ability to form a new hypothesis speeds dialog completion, and at higher error rates the gain in speed outweighs the per-turn penalty associated with taking the $ask()$ action repeatedly.

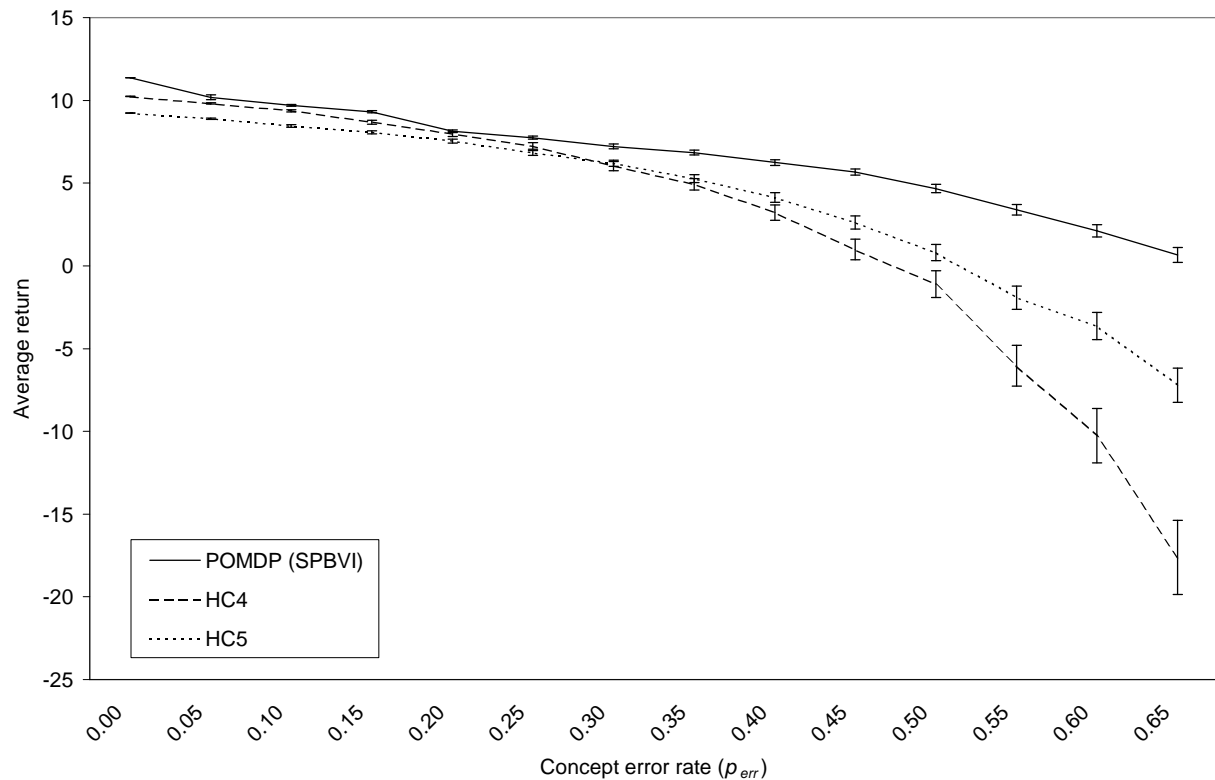


Figure 5.11 Concept error rate (p_{err}) vs. average return for SPBVI and Handcrafted controllers on the testing user model (one slot, no confidence score).

Assessing the scalability of SPBVI

Finally, the scalability of SPBVI was compared to direct optimization. First, a version of MAXITRAVEL was created which contained 10 user goals (instead of 100). PBVI (as implemented in *Perseus* [111]) was applied to this task with $N = 500$ belief points, and it was found that PBVI was not able to find a good policy. This finding illustrates how poorly PBVI scales for the MAXITRAVEL application, but it does not provide a sizeable operating range (over number of user goals) to compare PBVI and SPBVI. To address this, a simplified SDS-POMDP was created based on MAXITRAVEL, but in which the dialog history component \mathcal{S}_d was removed. The number of

user goals was set to $|\mathcal{S}_u| = C$, the error rate was set to $p_{err} = 0.30$, and optimizations were run with increasing values of C using SPBVI with $N = 100$ belief points and $K = 50$ observation samples, and PBVI with 500 belief points. Results are shown in Figure 5.12. For small problems, i.e., lower values of C , SPBVI performs equivalently to the PBVI baseline, but for larger problems, SPBVI outperforms PBVI by an increasing margin until $C = 100$ at which point PBVI was not able to find policies. Moreover, the SPBVI policies were computed using 80% fewer belief points than the baseline; i.e., the summary method’s policies scale to large problems and are much more compact.

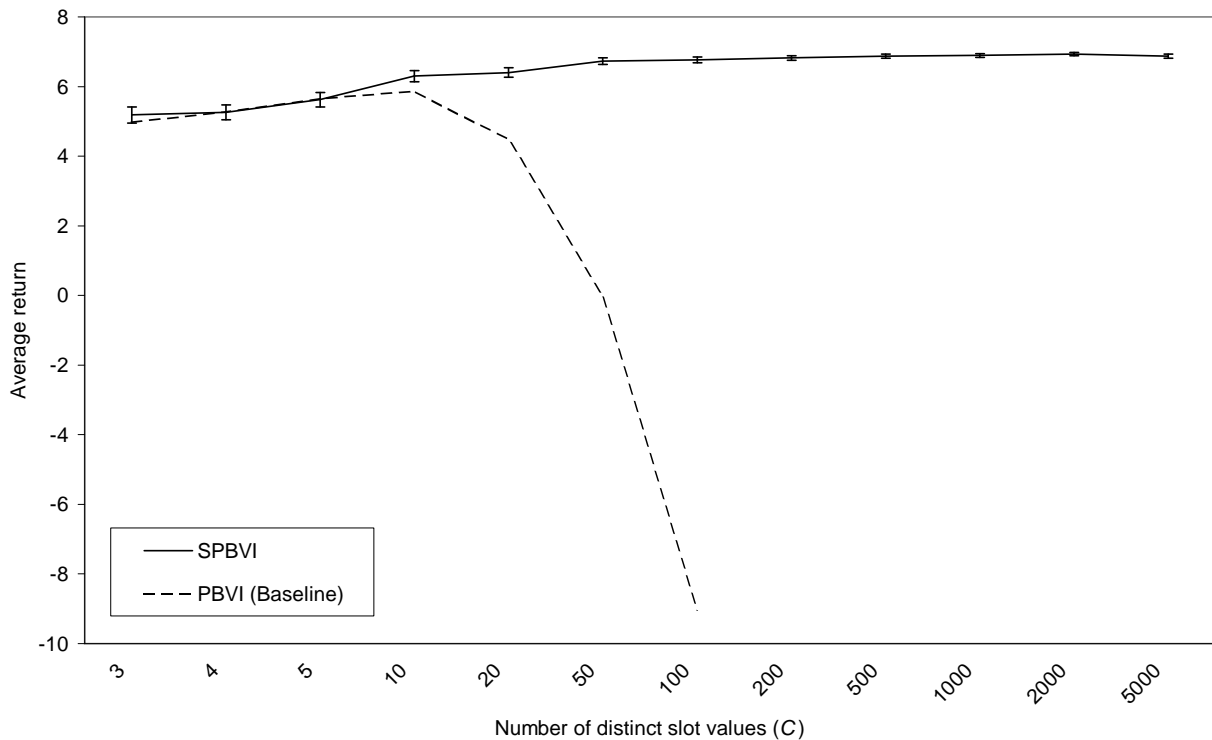


Figure 5.12 Number of distinct slot values (C) vs. average return for a simplified 1-slot dialog problem.

It is interesting to note that as C increases, the performance of the summary POMDP method appears to increase toward an asymptote. This trend is due to the fact that all confusions are equally likely in this model. For a given error rate, the more concepts in the model, the less likely consistent confusions are. Thus, having more concepts actually helps the policy identify spurious evidence over the course of a dialog. In practice of course the concept error rate p_{err} may increase as concepts are added.

The results shown above demonstrate that SPBVI effectively scales the SDS-POMDP model to handle many slot values, albeit limited to single-slot dialog applications. Although SPBVI faces several *potential* issues – i.e., sufficient coverage of belief space, lack of value gradient estimation, and Markov assumption violations – in practice SPBVI out-performs an MDP and two hand-crafted dialog managers while scaling to problem beyond the reach of other optimization techniques. The one-slot limitation is of course significant, and in the next Chapter SPBVI is extended to handle many slots.

Scaling up: Composite SPBVI (CSPBVI)

The previous chapter introduced a method – summary point-based value iteration (SPBVI) – for scaling a slot-filling SDS-POMDP to handle many slot values for a single slot. Real dialog systems consist of many slots, and this chapter tackles the problem of how the SDS-POMDP model can be scaled to handle many slots.

A straightforward application of SPBVI still grows exponentially with number of slots. Consider an SDS-POMDP with W slots. For clarity \mathcal{S}_u will again be restated as $\mathbb{S}_u = \{(s_u^1, \dots, s_u^W)\}$ where $s_u^w \in \mathcal{S}_u^w$ and where \mathcal{S}_u^w refers to the set of values for slot w . This gives rise to $|\mathbb{S}_u| = \prod_w |\mathcal{S}_u^w|$ distinct user goals, which grows exponentially with W . Summary space could then be constructed by mapping the belief mass of each slot $b(s_u^w)$ to $\hat{b}(\hat{s}_u^w)$ where $\hat{s}_u^w \in \{best, rest\}$. Although SPBVI reduces the dimensionality of the planning simplex to $\prod_w |\hat{\mathcal{S}}_u^w| = 2^W$ user goals, this still grows exponentially in W , and in practice would still be limited to applications with a small handful of slots.

The intuition of *composite* SPBVI (CSPBVI) is rather to create *many* simple policies: a distinct, manageably-sized summary space is created for each slot, and optimization for each slot is performed separately. This process creates W policies. At runtime, actions are selected by combining these W policies into a single “composite policy” using a simple heuristic. This chapter presents the CSPBVI method in detail, evaluates it on an extended version of the MAXITRAVEL application called MAXITRAVEL- W , and finally describes a working spoken dialog system based on CSPBVI.

6.1 CSPBVI method description

As with SPBVI, CSPBVI consists of four phases: construction, sampling, optimization, and execution. In the construction phase, $W + 1$ POMDPs are created. The first of these, again called the *master* POMDP, is an SDS-POMDP with several constraints and additions. In the master POMDP, the user’s goal is written \mathbb{S}_u and is decomposed into W slots, $\mathbb{S}_u = \{(s_u^1, \dots, s_u^W)\}$ where $s_u^w \in \mathcal{S}_u^w$ and where \mathcal{S}_u^w refers to the set of values for slot w . The dialog history is similarly written \mathbb{S}_d and is decomposed into W slots, $\mathbb{S}_d = \{(s_d^1, \dots, s_d^W)\}$ where $s_d^w \in \mathcal{S}_d^w$ and where \mathcal{S}_d^w refers to the set

of possible dialog histories for w . It is assumed that the number of possible dialog histories for each slot $|\mathcal{S}_d^w|$ is constant with respect to the number of slots W , and with respect to the number of user goals for each slot $|\mathcal{S}_u^w|$.

In the master POMDP, machine actions are formed of predicates which take arguments that encode the slot w and the value (or values) s_u^w which the action refers to. Machine actions are written $predicate[w](s_u^w)$, where $predicate$ refers to the illocutionary force of the action, w refers to a slot index, and s_u^w refers to the slot value(s) referred to by the action, if any. For example, the SDS-POMDP machine actions *ask-from* and *confirm-to-london* would be restated as $ask[from]()$ and $confirm[to](london)$. A special meta-slot $w = all$ denotes an action that refers to all slots, such as $submit[all](s_u^{from} = london, s_u^{to} = cambridge)$ and $greet[all]()$.

Finally, in the master POMDP a modified reward function is created $r_w(s, a_m)$ which removes conditioning on all but the slot w . For example, if the reward r for incorrectly/correctly submitting a user's goal is $-25/+25$, then r_w would assess $-25/+25$ for incorrectly/correctly submitting slot w , ignoring all others.

After the master POMDP is formed, W summary POMDPs are constructed. Each of these has a state space with two components, $\hat{\mathcal{S}}_u^w$ and $\hat{\mathcal{S}}_d^w$, where $\hat{\mathcal{S}}_u^w = \{best, rest\}$ and $\hat{\mathcal{S}}_d^w = \mathcal{S}_d^w$. The action set of each of these summary POMDPs consists of the predicates of \mathcal{A}_m and take one argument, $\hat{w} \in \{this, other\}$, where $\hat{w} = this$ indicates that an action in master space a_m refers to *this* slot and $\hat{w} = other$ indicates that a_m refers to *some other* slot. (If the action a_m operates on *all* slots, \hat{w} is set to *this*.) For example, a master POMDP action $a_m = confirm[from](london)$ would be mapped to $\hat{a}_m^{from} = confirm[this]$ in the summary POMDP for the *from* slot, and $\hat{a}_m^{to} = confirm[other]$ in the summary POMDP for the *to* slot. Actions can also be mapped from summary space into master space using function $aToMasterComposite$ (Algorithm 11), which sets the argument of the action a_m to be the most likely user goal in $b(s_u^w)$. If the summary action takes the argument $\hat{w} = this$, then the action in master space is set to the current slot w ; if the summary action takes the argument $\hat{w} = other$ then another slot w is chosen *at random*. This latter usage of $aToMasterComposite$ is only used during the sampling phase to explore belief space, and isn't needed at runtime. Finally, another function $bToSummaryComposite$ (Algorithm 12) maps from a given belief point in master space b and selected summary POMDP w to a belief point in that summary POMDP \hat{b} .

In the *sampling* phase of CSPBVI (Algorithm 13), points are sampled for *each* summary POMDP. For each slot w three quantities are produced: a set of points $\{\hat{b}_{w,n}\}$ encountered by following a random policy, a set of slot-specific rewards (as described above) $\{\hat{r}_{w,n}^{\hat{a},k}\}$, and a set of indexes $\{l(w, n, \hat{a}_m, k)\}$ which estimates $SE(w, \hat{b}, \hat{a}_m, k)$ by giving the index n of the closest point in $\{\hat{b}_{w,n}\}$ when action \hat{a}_m is taken and observation index k is received.¹ Summary actions which act on both $\hat{w} = this$ and $\hat{w} = other$ are taken, which ensures that belief points resulting from taking actions on both this slot and on other slots are encountered. As desired, the *dimensionality* of each summary POMDP remains constant as the number of slots grows, and W

¹As in the previous chapter, when a_m or \hat{a}_m is written in a subscript or superscript, it will be shortened to a or \hat{a} , as in x_a or $x_{\hat{a}}$.

Algorithm 11: Function *aToMasterComposite*.

Input: \hat{a}_m, \hat{w}, w, b
Output: a_m

- 1 **if** \hat{a}_m operates on all slots **then**
- 2 **for** $\tilde{w} \leftarrow 1$ **to** W **do**
- 3 $s_u^{\tilde{w}} \leftarrow \arg \max_{\tilde{s}_u^{\tilde{w}}} b(\tilde{s}_u^{\tilde{w}})$
- 4 $a_m \leftarrow \hat{a}_m[\text{all}](s_u^1, \dots, s_u^W)$
- 5 **else**
- 6 **if** $\hat{w} = \text{this}$ **then**
- 7 $w^* = w$
- 8 **else**
- 9 $w^* = \text{randIntOmit}(W, w)$
- 10 **if** $\hat{a}_m[w^*]$ takes an argument in master space **then**
- 11 $s_u^{w^*} \leftarrow \arg \max_{s_u^{w^*}} b(s_u^{w^*})$
- 12 $a_m \leftarrow \hat{a}_m[w^*](s_u^{w^*})$
- 13 **else**
- 14 $a_m \leftarrow \hat{a}_m[w^*]()$

Algorithm 12: Function *bToSummaryComposite*.

Input: b, w
Output: \hat{b} (for slot w)

- 1 $\hat{b}(\hat{s}_u = \text{best}) \leftarrow \max_{s_u^w} b(s_u^w)$
- 2 $\hat{b}(\hat{s}_u = \text{rest}) \leftarrow 1 - \hat{b}(\hat{s}_u = \text{best})$
- 3 **foreach** $s_d^w \in \mathcal{S}_d^w$ **do**
- 4 $\hat{b}(s_d^w) \leftarrow b(s_d^w)$

optimizations are required, one for each slot.

Optimization in CSPBVI, shown in Algorithm 15, is similar to optimization in SPBVI (Algorithm 9, page 72) with two important differences. First, CSPBVI optimization is run W times, once for each slot w using that slot's dynamics and reward given in $\{\hat{b}_{w,n}\}$, $\{\hat{r}_{w,n}^k\}$, and $\{l(w, n, \hat{a}_m, k)\}$. Second, summary actions selected in each iteration are *restricted* to $\hat{w} = \text{this}$: that is, only actions which operate on *this* slot (or all slots) are incorporated into conditional plans. In other words, the policy for a slot will only contain actions which operate on itself. This is an important design decision which prevents a policy from suggesting that some other slot ought to take an action. Optimization produces an optimal summary action $\hat{a}^{w,n}$ where $\hat{a}^{w,n} \in \hat{\mathcal{A}}_m$ for each point $\hat{b}_{w,n}$.

Policy execution proceeds as Algorithm 16. Belief monitoring is performed in the master POMDP, actions are *nominated* by each of the summary POMDPs, and a simple heuristic chooses

Algorithm 13: Belief point selection for CSPBVI.

```

Input:  $\mathfrak{P}_{SDS}, \epsilon, N, K, W$ 
Output:  $\{\hat{b}_{w,n}\}, \{\hat{r}_{w,n}^{a,k}\}, \{l(w, n, \hat{a}_m, k)\}$ 
// Iterate over each slot  $w$ .
1 for  $w \leftarrow 1$  to  $W$  do
    // First, sample points using a random policy
2      $s \leftarrow \text{sampleDist}_s(b)$ 
3      $n \leftarrow 1$ 
4      $b \leftarrow b_0$ 
5      $\hat{b}_{w,n} \leftarrow \text{bToSummaryComposite}(b, w)$ 
6      $(\hat{r}_{w,n}^{a,k}, \hat{b}_{w,n}^{a,k}) \leftarrow \text{samplePointComposite}(\mathfrak{P}_{SDS}, b, w, K)$ 
7     while  $n < N$  do
        // Take a random action and compute new belief state.
8          $\hat{a}_m \leftarrow \text{randElem}(|\hat{\mathcal{A}}_m|)$ 
9          $\hat{w} \leftarrow \text{randElem}(\{this, other\})$ 
10         $a_m \leftarrow \text{aToMasterComposite}(\hat{a}_m, \hat{w}, w, b)$ 
11         $s' \leftarrow \text{sampleDist}_{s'}(P(s'|s, a_m))$ 
12         $o' \leftarrow \text{sampleDist}_{o'}(P(o'|s', a_m))$ 
13         $b \leftarrow SE(b, a_m, o')$ 
14         $\hat{b} \leftarrow \text{bToSummaryComposite}(b, w)$ 
        // If this is a (sufficiently) new point, add it to  $\hat{\mathcal{B}}$ .
15        if  $\min_{i \in [1, n]} |\hat{b}_{w,i} - \hat{b}| > \epsilon$  then
16             $n \leftarrow (n + 1)$ 
17             $\hat{b}_{w,n} \leftarrow \hat{b}$ 
18             $(\{\hat{r}_{w,n}^{a,k}\}, \{\hat{b}_{w,n}^{a,k}\}) \leftarrow \text{samplePointComposite}(\mathfrak{P}_{SDS}, b, w, K)$ 
19             $s \leftarrow s'$ 
        // Second, sample corners  $\hat{s}$  of summary space
20        foreach  $\hat{s} \in \hat{\mathcal{S}}_w$  do
21            foreach  $\tilde{s} \in \hat{\mathcal{S}}_w$  do
22                 $\hat{b}(\tilde{s}) \leftarrow 0$ 
23             $\hat{b}(\hat{s}) \leftarrow 1$ 
24            if  $\min_{i \in [1, n]} |\hat{b}_{w,i} - \hat{b}| > \epsilon$  then
25                 $n \leftarrow (n + 1); N \leftarrow n$ 
26                 $b \leftarrow \text{sampleCorner}(b_0, \hat{s})$ 
27                 $\hat{b}_{w,n} \leftarrow \text{bToSummaryComposite}(b, w)$ 
28                 $(\{\hat{r}_{w,n}^{a,k}\}, \{\hat{b}_{w,n}^{a,k}\}) \leftarrow \text{samplePointComposite}(\mathfrak{P}_{SDS}, b, w, K)$ 
        // Finally, find index  $l(w, n, \hat{a}_m, k)$  of closest point to  $b_{w,n}^{a,k}$ 
29        for  $n \leftarrow 1$  to  $N$  do
30            foreach  $\hat{a}_m \in \hat{\mathcal{A}}_m^w$  do
31                for  $k \leftarrow 1$  to  $K$  do
32                     $l(w, n, \hat{a}_m, k) \leftarrow \arg \min_{\tilde{n}} |b_{w,n}^{a,k} - b_{w,\tilde{n}}|$ 

```

Algorithm 14: Function *samplePointComposite*.

Input: $\mathfrak{P}_{SDS}, b, w, K$
Output: $\{\hat{b}^{\hat{a},k}\}, \{\hat{r}^{\hat{a},k}\}$

- 1 **foreach** $\hat{a}_m \in \hat{\mathcal{A}}_m^w$ **do**
- 2 **foreach** $\hat{w} \in \hat{\mathcal{W}}$ **do**
- 3 **for** $k \leftarrow 1$ **to** K **do**
- 4 // Sample a (possibly) new master state \tilde{s} .
- 4 $\tilde{s} \leftarrow \text{sampleDist}_{\tilde{s}}(b)$
- 5 $a_m \leftarrow \text{aToMasterComposite}(\hat{a}_m, \hat{w}, w, b)$
- 6 $s' \leftarrow \text{sampleDist}_{s'}(P(s'|\tilde{s}, a_m))$
- 7 $o' \leftarrow \text{sampleDist}_{o'}(P(o'|s', a_m))$
- 8 // Compute the successor master state b' and save $\hat{r}^{\hat{a}[\hat{w}],k}$ and $\hat{b}^{\hat{a}[\hat{w}],k}$.
- 8 $b' \leftarrow SE(b, a_m, o')$
- 9 $\hat{r}^{\hat{a}[\hat{w}],k} \leftarrow r_w(\tilde{s}, a_m)$
- 10 $\hat{b}^{\hat{a}[\hat{w}],k} \leftarrow \text{bToSummaryComposite}(b', w)$

Algorithm 15: CSPBVI optimization procedure.

Input: $\mathfrak{P}_{SDS}, \{\hat{b}_{w,n}\}, \{l(w, n, \hat{a}_m, k)\}, \{\hat{r}_{w,n}^{\hat{a},k}\}, K, T$
Output: $\{\hat{a}_t^{w,n}\}$

- 1 **for** $w \leftarrow 1$ **to** W **do**
- 2 $N \leftarrow |\{\hat{b}_{w,n}\}|$
- 3 **for** $n \leftarrow 1$ **to** N **do**
- 4 $\hat{v}_0^n \leftarrow 0$
- 5 **for** $t \leftarrow 1$ **to** T **do**
- 6 // Generate $\{\hat{v}^{\hat{a},n}\}$, values of all possibly useful CPs.
- 6 **for** $n \leftarrow 1$ **to** N **do**
- 7 **foreach** $\hat{a}_m \in \hat{\mathcal{A}}_m^w$ **do**
- 8 $\hat{v}^{\hat{a},n} \leftarrow \left(\frac{1}{K} \sum_k \hat{r}_{w,n}^{\hat{a}[\hat{w}=\text{this},k]} \right) + \left(\frac{\gamma}{K} \sum_k \hat{v}_{t-1}^{l(w,n,\hat{a}[\hat{w}=\text{this},k])} \right)$
- 9 // Prune $\{\hat{v}^{\hat{a},n}\}$ to yield $\{\hat{v}_t^n\}$, values of actually useful CPs.
- 9 **for** $n \leftarrow 1$ **to** N **do**
- 10 $\hat{a}^* \leftarrow \arg \max_{\hat{a}} \hat{v}^{\hat{a},n}$
- 11 $\hat{a}_t^{w,n} \leftarrow \hat{a}^*$
- 12 $\hat{v}_t^n \leftarrow \hat{v}^{\hat{a}^*,n}$

which of these nominated actions to take. For a given belief point b , the corresponding set of summary belief points $\hat{b}_w = bToSummaryComposite(b, w)$ is computed for all slots w . For each belief point \hat{b}_w the index of the closest point n^* in the set $\{\hat{b}_{w,n}\}$ is found, and its summary action (\hat{a}^{w,n^*}) is mapped to a master action $a_m^w = aToMaster(\hat{a}^{w,n^*}, \hat{w} = this, w, b)$. This process is repeated for each slot w and produces a vector of nominated master actions, a_m^w . Finally, a handcrafted heuristic called *chooseActionHeuristic* (created for each application) selects an action from this vector to take. For example, a heuristic might first look for an *ask* action, then look for a *confirm* action, then finally look for a *submit* action. In practice, belief monitoring in a very large SDS-POMDP may require approximation; although not specified here explicitly, the next section provides an example of how this can be done.

Algorithm 16: CSPBVI action selection procedure, used at runtime.

Input: $b, \{\hat{b}_{w,n}\}, \{\hat{a}_{w,n}\}$

Output: a_m

```

1 for  $w \leftarrow 1$  to  $W$  do
2    $\hat{b} \leftarrow bToSummaryComposite(b, w)$ 
3    $n^* \leftarrow \arg \min_n |\hat{b}_{w,n} - \hat{b}|$ 
4    $a_m^w \leftarrow aToMasterComposite(\hat{a}^{w,n^*}, \hat{w} = this, b, w)$ 
5  $a_m \leftarrow chooseActionHeuristic(\{a_m^w\})$ 

```

In sum, CSPBVI scales to many slots by estimating system dynamics from the standpoint of each slot *myopically*: detailed estimates are made for actions which refer to *this* slot, but gross estimates are made for actions which refer to *all other* slots. Optimization is then performed using these myopic system dynamics, producing a mini dialog manager for each slot. At runtime, each mini dialog manager nominates an action, and a handcrafted heuristic chooses one action from these. The success of CSPBVI relies on its assumptions that a “myopic” view of system dynamics is sufficient for planning, and that a handcrafted heuristic can successfully choose actions. The next two sections of this chapter tests these assumptions through dialog simulation experiments.

6.2 Example CSPBVI application: MAXITRAVEL-W

To test CSPBVI, MAXITRAVEL-W was created. MAXITRAVEL-W is an SDS-POMDP with w slots. To keep belief monitoring tractable, some independence assumptions between slots are needed and the approach taken here is to compose an SDS-POMDP from W instances of MAXITRAVEL (Section 5.4, (page 84), extended in several respects. First, user actions are decomposed by slot into $a_u = (a_u^1, \dots, a_u^W)$, and each per-slot user action element a_u^w is decomposed into three components $a_u^w = (a_{state}^w, a_{stateSlot}^w, a_{yesNo}^w)$, where $a_{state}^w \in \mathcal{A}_{state}^w$, $a_{stateSlot}^w \in \mathcal{A}_{stateSlot}^w$, and $a_{yesNo}^w \in \mathcal{A}_{yesNo}^w$. \mathcal{A}_{state}^w consists of $state[w](s_u^w)$ and indicates the user said their goal *without identifying* which slot it corresponds to – for example, “London” or “10:00 AM”. $\mathcal{A}_{stateSlot}^w$ consists of $stateSlot[w](s_u^w)$

and indicates the user said their goal *and identified* which slot it corresponds to – for example, “to London”, “from London”, “leaving at 10:00 AM”, or “arriving at 10:00 AM”. Finally $\mathcal{A}_{\text{yesNo}}^w$ includes actions *yes* and *no*. The sets $\mathcal{A}_{\text{state}}^w$, $\mathcal{A}_{\text{stateSlot}}^w$, and $\mathcal{A}_{\text{yesNo}}^w$ each also contain *null*. This formulation enables rather complex actions to be formed: see Table 6.1 for examples.

Next, the user action model $p(a'_u | s'_u, s_d, a_m)$ was extended to support this formulation of \mathcal{A}_u . Each slot contains a slot-specific user model, conditioned on whether the machine is asking about *this* slot or *another* slot. To estimate the user model parameters, additional annotation of the SACTI-1 data [125] was performed. As described above, the corpus was divided in half and two user models were created, called the *training* and *testing* models, shown in Table 6.2. The *training* model (only) is used in all results in this chapter, except the last experiment which also uses the *testing* model. Due to data sparsity in the SACTI-1 corpus, and as was done for MAXITRAVEL, the user acts *yes* and *no* were grouped into one class, so statistics for *yes* and *no* are equal (with appropriate conditioning for the sense *yes* vs. *no*). In addition, as required by CSPBVI, the reward function was factored per-slot into $r_w(s, a)$ such that actions performed on *this* slot received small per-turn rewards as listed in Table 5.5 (page 76), and actions performed on *other* slots receive 0. The submit action receives $12.5 \cdot W$ if the user’s goal is satisfied entirely correctly and $-12.5 \cdot W$ otherwise (where W is the number of slots).

Finally, the observation model was changed significantly. First the observation model was separated into a *generation* model and an *inference* model. The generation model attempts to realistically simulate speech recognition errors by making concept confusions with probability p_{err} , where a confusion changes a non-null user action component to any other (possibly null) component. For example, if one concept error is made, the user action “Yes, London” might be changed to “Frankfurt London” or even “Yes No”. Since *null* is one type of user action, the generation model also simulates deletion errors – for example, “Yes, London” could be changed to “Yes”, “London” or *null*. The model does not simulate insertion errors. Also, each observation component (such as “London” or “To Edinburgh” or “Yes”) carries with it a *per-concept* confidence score. Confidence scores for correct recognitions are sampled from $p_h(c)$ and incorrect recognitions are sampled from $p_h(1 - c)$, where $p_h(c)$ is given by:²

$$p_h(c) = \frac{he^{hc}}{e^h - 1}. \quad (6.1)$$

Ideally the observation inference model should express the probability of the entire observation given the entire user action $P(\text{whole observation} | a_u^1, \dots, a_u^W)$, but this formulation would complicate belief monitoring significantly by requiring a joint distribution over all user goals to be computed. Instead, the observation model estimates $P(\text{whole observation} | a_u^{w'})$ *separately* for each slot:

$$P(\tilde{a}'_u, c' | a_u^{w'}) \approx \begin{cases} p_h(c'_i) \cdot (1 - p_{err}) & \text{if there exists an observation component} \\ & i \text{ in } \tilde{a}'_u \text{ with the same type as } a_u^{w'}, \\ \frac{p_{err}}{|\mathcal{A}_u^w| - 1} & \text{otherwise,} \end{cases} \quad (6.2)$$

²Reprinted here from Equation 3.16 on page 33 for ease of reference.

Machine Action		Example user response	
Utterance	a_m	Utterance	$(a_{state}^{from}, a_{stateSlot}^{from}, a_{yesNo}^{from}, a_{state}^{to}, a_{stateSlot}^{to}, a_{yesNo}^{to})$
“Where are you going to?”	ask[to]()	“London”	(-, -, -, london, -, -)
		“To London”	(-, -, -, -, london, -)
		“London from Cambridge”	(-, cambridge, -, london, -, -)
		“To London from Cambridge”	(-, cambridge, -, -, london, -)
“To London, is that right?”	confirm[to](london)	“Yes”	(-, -, -, -, -, yes)
		“Yes, to London”	(-, -, -, -, london, yes)
“From Oxford, is that right?”	confirm[from](oxford)	“Yes, to London from Cambridge”	(-, cambridge, -, -, london, yes)
		“No”	(-, -, -, -, -, -)
		“No, Cambridge”	(cambridge, -, no, -, -, -)

Table 6.1 Example machine actions and user responses used in MAXITRAVEL-W. The dash character “-” stands for null.

Machine Action		User Response (to slot only)		$P(a_u^{to'} s_u^{to'} = \text{London}, a_m)$	
Utterance	a_m	Utterance	$a_u^{to'} = (a_{state}^{to'}, a_{stateSlot}^{to'}, a_{yesNo}^{to'})$	Training	Testing
“Where are you going to?”	<code>ask[to]()</code>	“London”	<code>(london, -, -)</code>	0.521	0.532
		“To London”	<code>(-, london, -)</code>	0.467	0.443
“Where are you leaving from?”	<code>ask[from]()</code>	—	<code>(-, -, -)</code>	0.013	0.025
		“To London”	<code>(-, london, -)</code>	0.146	0.212
“To London, is that right?”	<code>confirm[to](london)</code>	—	<code>(-, -, -)</code>	0.855	0.788
		“Yes”	<code>(-, -, yes)</code>	0.782	0.806
“To London, is that right?”	<code>confirm[to](london)</code>	“Yes, London”	<code>(london, -, yes)</code>	0.093	0.042
		“Yes, to London”	<code>(-, london, yes)</code>	0.112	0.127
“To Edinburgh, is that right?”	<code>confirm[to](edinburgh)</code>	—	<code>(-, -, -)</code>	0.013	0.025
		“No”	<code>(-, -, no)</code>	0.782	0.806
“To Edinburgh, is that right?”	<code>confirm[to](edinburgh)</code>	“No, London”	<code>(london, -, no)</code>	0.093	0.042
		“No, to London”	<code>(-, london, no)</code>	0.112	0.127
“From Oxford, is that right?”	<code>confirm[from](oxford)</code>	—	<code>(-, -, -)</code>	0.013	0.025
		“To London”	<code>(-, london, -)</code>	0.245	0.522
		—	<code>(-, -, -)</code>	0.755	0.478

Table 6.2 Summary of user model parameters for the MAXITRAVEL-W application. Dialog data from the SACTI-1 corpus was annotated and split in two; half was used to estimate the training user model and half for the testing user model. The dash character (—) indicates null.

where example *types* include “place names”, “dates”, or “boolean” (yes/no).

As described above, CSPBVI requires an application-dependent heuristic called *chooseActionHeuristic* to choose among actions nominated by each slot. For the MAXITRAVEL-W application, this heuristic is shown in Algorithm 17 and first looks for an *ask* action by considering the slots in order; if it doesn’t find one, it then looks for a *confirm* action again considering the slots in order; and if it doesn’t find one, then all slots must have nominated the *submit* action, which is selected.

Algorithm 17: Function *chooseActionHeuristic* for the MAXITRAVEL-W SDS-POMDP.

```

Input:  $\{a_m^w\}$ 
Output:  $a_m$ 
// First look for an ‘ask’ action, considering slots in order.
1 for  $w \leftarrow 1$  to  $W$  do
2   if  $a_m^w = ask$  then
3      $a_m \leftarrow a_m^w$ 
4     return
// If not found, look for a ‘confirm’ action.
5 for  $w \leftarrow 1$  to  $W$  do
6   if  $a_m^w = confirm$  then
7      $a_m \leftarrow a_m^w$ 
8     return
// If not found, then by definition, all slots
// have nominated ‘submit’, so return this.
9  $a_m \leftarrow a_m^1$ 

```

Like SPBVI, the CSPBVI optimization procedure takes a set of parameters K , N , ϵ , and T . Since the POMDP for each slot in the MAXITRAVEL-W application is of approximately the same size and complexity as the MAXITRAVEL application, it seems reasonable that similar choices for these parameters would be suitable (i.e., $N = 100, K = 50, T = 50$). Experimentation (not shown here) confirmed this, and all experiments in this chapter use these values.

Figures 6.1 and 6.2 together show one conversation with a 2-slot version of MAXITRAVEL-W with a reasonable concept error rate and a somewhat informative confidence score ($p_{err} = 0.30, h = 2$). In this example, the user is trying to buy a ticket from London to Cambridge.

Initially belief mass is spread over all user goals evenly, and at the beginning of the conversation in Figure 6.1, the summary belief state for each slot is passed to its respective policy, and both the *from* slot and the *to* slot nominate the *ask* action. The heuristic examines these two actions and selects the *ask[from]()* action (M1). The user’s response (U1) of “London” is misrecognized as “Leeds” with relatively high confidence (0.67), causing a large shift in belief mass toward *leeds* in the *from* slot, and no change to the belief state of the *to* slot. For the second ma-

chine action (M2), the *from* slot nominates the *confirm* action and the *to* slot nominates the *ask* action. The heuristic examines these and selects the *ask[to]()* action (M2). The user’s response (U2) of “To Cambridge” is again mis-recognized, this time as “From Oxford”, with moderate confidence (0.45). However, since the user model indicates this is a relatively unlikely user action, and because the confidence score is somewhat low, this results in only a moderate shift of belief mass toward *oxford* in the *from* slot (and again no change to the *to* slot). For the third machine action (M3), the *from* slot again nominates the *confirm* action and the *to* slot nominates the *ask* action; the heuristic examines these and again selects the *ask[to]()* action (M3).

Continuing to Figure 6.2, the user’s response to M3 (U3) of “To Cambridge” is correctly recognized with very high confidence (0.97). This causes a massive shift of belief mass toward *cambridge* in the *to* slot (and no change to the *from* slot). For the fourth machine action (M4), the *from* slot is in the same belief state and again nominates the *confirm* action. The *to* slot is now in a belief state with high certainty, and nominates the *submit* action. The heuristic examines these and selects the *confirm[from](leeds)* action (M2). To this the user responds “No, from London” (U4) but this is mis-recognized as “No, to Edinburgh”. The *no* portion of the response is recognized with high confidence (0.78) and is also highly predicted by the user model, so as a result a great deal of belief mass moves away from *leeds* in the *from* slot. However, the “to Edinburgh” portion is predicted as highly unlikely by the user model, and is also recognized with very low confidence (0.13), and has virtually no effect on the *to* slot. For the fifth machine action (M5), the *from* slot is now in a belief state with relatively low certainty, and nominates the *ask* action. The *to* slot is still in a state of very high certainty and again nominates the *submit* action. Between these the heuristic selects the “ask[from]()” action (M5), and the user’s response of “From London to Cambridge” (U5) is correctly understood. The “From London” portion is recognized with very high confidence (0.95) and causes a massive shift in belief mass; the “to Cambridge” portion is recognized with relatively low confidence (0.33), and as a result has only a marginal effect on the *to* slot. For the sixth machine action, both slots are in a state of high certainty and both nominate the *submit* action; the heuristic chooses $a_m = submit[all](s_u^{from} = london, s_u^{to} = cambridge)$ and finished the dialog successfully.

6.3 Comparisons with baselines

CSPBVI assume that *local* planning within each slot combined with a heuristic will provide a good approximation to *global* planning over all slots. To test this assumption, a baseline optimization method which performs global planning over all slots was created by extending SPBVI as described in the beginning of this chapter: i.e., one summary space was formed which contained a component $\mathcal{S}_u^w = \{best, rest\}$ for each slot w , resulting in $\prod_w |\hat{\mathcal{S}}_u^w| = 2^W$ user goals in summary space. A 2-slot version of MAXITRAVEL-W was created with 1000 goals/slot and optimized with CSPBVI and the SPBVI baseline. CSPBVI was applied as described above, with $N = 100$ belief points and $K = 50$ observation samples per belief point, and SPBVI was applied with $N = 500$ belief points and $K = 50$ observation samples per belief point. Results are shown in Figure 6.3.

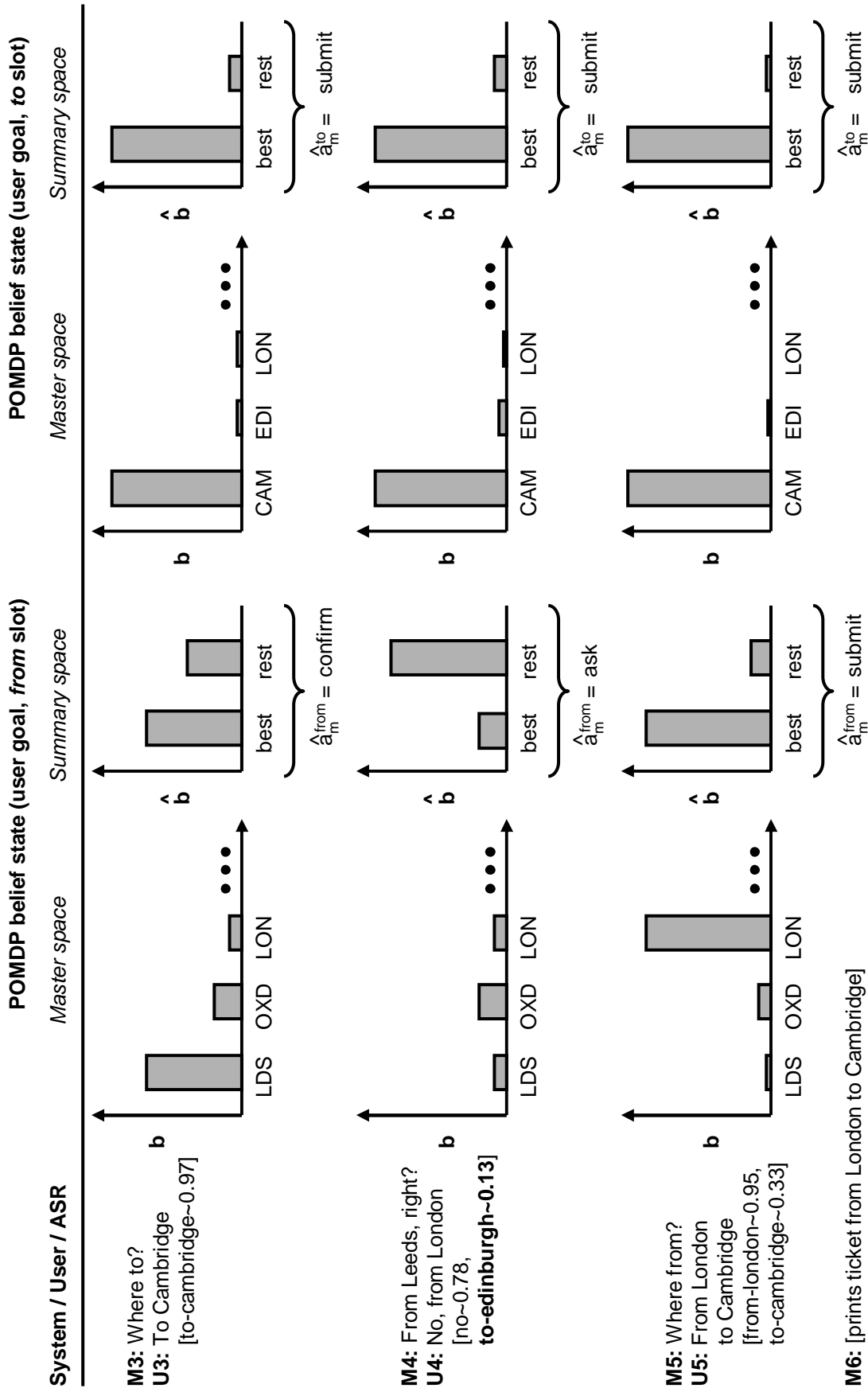


Figure 6.2 Sample conversation between user and CSPBVI-based dialog manager (second half, continued from Figure 6.1). LDS stands for “Leeds”, OXD stands for “Oxford”, LON stands for “London”, CAM stands for “Cambridge”, and EDI stands for “Edinburgh”. a_m^{from} and a_m^{to} indicate the actions nominated by each slot.

When there are no recognition errors (i.e., when $p_{err} = 0$) the two methods perform the same; however, as errors increase, CSPBVI performs marginally worse than the SPBVI baseline. This indicates that the CSPBVI assumptions are indeed having an impact on solution quality. That said, the magnitude of performance loss is small in comparison to the effects of concept error rate p_{err} : the impact of using CSPBVI instead of SPBVI is generally equivalent to less than a 5% increase in concept error rate (p_{err}) or less.

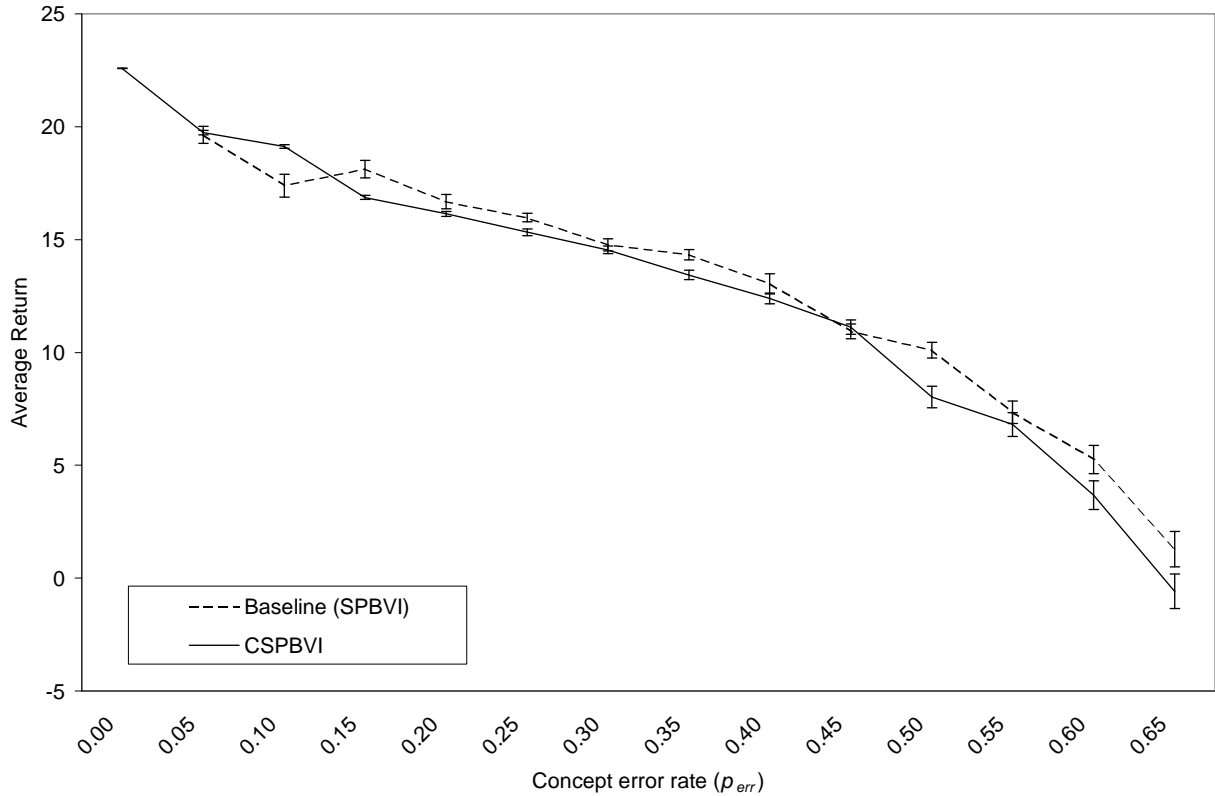


Figure 6.3 Concept error rate (p_{err}) vs. average return for CSPBVI and SPBVI baseline for MAXITRAVEL-W with 2 slots.

This experiment was attempted for $w = 3$ but it was found that SPBVI becomes intractable for $w > 2$. This is not surprising given that number of summary actions $|\hat{\mathcal{A}}_m|$ grows with the addition of each slot, causing the number of sampled “successor” points $|\{\hat{b}_n^{\hat{a},k}\}|$ to explode.

The performance of the MAXITRAVEL-W application optimized with CSPBVI was then compared to the same set of baselines used earlier in this thesis: a comparison with an MDP *excluding* confidence score quantifies the benefit of maintaining multiple hypotheses; a comparison with an MDP *with* confidence score tests how well confidence score information is exploited; and a comparison with a handcrafted controllers assess the overall benefits of automated planning.

CSPBVI was first compared to an MDP with no contribution of confidence score information (i.e., with $h = 0$). MAXITRAVEL-W with 2 slots was created, where each slot contained 100 values (for each slot). CSPBVI was run using $N = 100$ belief points and $K = 50$ observation samples. Two MDP baselines were created: the first baseline, called “MDP-Full”, used the same “flat” state space as described in Table 4.2 (page 48), which enumerates all possible observed dialog

states, resulting in 11 MDP states. The second baseline, called “MDP-Composite”, estimated a separate MDP policy for each slot, and combined actions together into a “composite” action using the same heuristic as CSPBVI. This second baseline consisted of two separate MDPs, each with 5 states (i.e., *dialog-start*, *n*, *u*, *c*, and *dialog-end*). Both MDP baselines were trained using Q-learning.

Results are shown in Figure 6.4. The Y-axis shows average reward gained per dialog, and error bars indicate the 95% confidence interval. When no recognition errors are made (i.e., $p_{err} = 0$), both MDP baselines and CSPBVI perform equivalently, again demonstrating that in the presence of perfect recognition, there is no value in maintaining multiple hypotheses. However as concept recognition errors become more common (i.e., as p_{err} increases), the POMDP approach outperforms the MDP approach by an increasing margin. Stated alternatively, compared to the MDP, the POMDP can “recover” much of the degradation caused by speech recognition errors: for example, the POMDP performance at a concept error rate of 40% matches the MDP performance at about 30%, and the POMDP performance at a concept error rate of 50% matches the MDP performance at about 35%.

This trend indicates that the assumptions introduced in the CSPBVI method (i.e., local optimization combined with a handcrafted heuristic) enables policy optimization to scale without forfeiting the ability of the POMDP to effectively exploit uncertainty in the dialog state. Also, MDP-Full appears to outperform MDP-Composite, indicating that an MDP with a richer state space which makes fewer independence assumptions is capable of attaining more reward.

This experiment was repeated for $w = 1 \dots 5$ at a variety of error rates using the same optimization parameters. Results are shown in Figure 6.5. The X-axis shows the number of slots, and the Y-axis shows the average return gained per dialog *per slot*. When no recognition errors are made (i.e., $p_{err} = 0.00$), the POMDP and MDPs perform identically but where concept recognition errors are made (i.e., $p_{err} > 0$), the POMDP outperforms the MDP. As the number of slots increases, in general average return declines slightly for all techniques, reflecting the fact that eliciting values for more slots results in longer dialogs. Where the concept error rate is relatively high ($p_{err} = 0.50$), performance of the POMDP is relatively stable with respect to number of slots, but performance of the MDP-based controllers varies. At 1 slot, MDP-Full and MDP-Composite are identical formulations so their performance is of course equal. At 2 slots, MDP-Full gains more reward than MDP-Composite. This appears to be because the MDP-Full state space is larger and richer enabling MDP-Full to construct a more detailed policy. However for 3, 4, and 5 slots, the performance of MDP-Full declines below that of MDP-Composite. This trend appears to be the result of under-training – the state space size for MDP-Full increases dramatically but the number of training runs was held constant (at 100,000 dialogs). The performance of MDP-Composite is relatively constant as its size is not increasing.

Confidence score information was next added. The “MDP-2” baseline was used as presented in section 4.2 (page 48) formulated as a composite policy as described above.³ Figures 6.6, 6.7,

³Optimization on a “full” MDP-2 model proved infeasible because for W slots, there are $7^W + 2$ states which grows exponentially in W . For example, for the 5-slot problem, a full state space for MDP-2 contains 16,809 states.

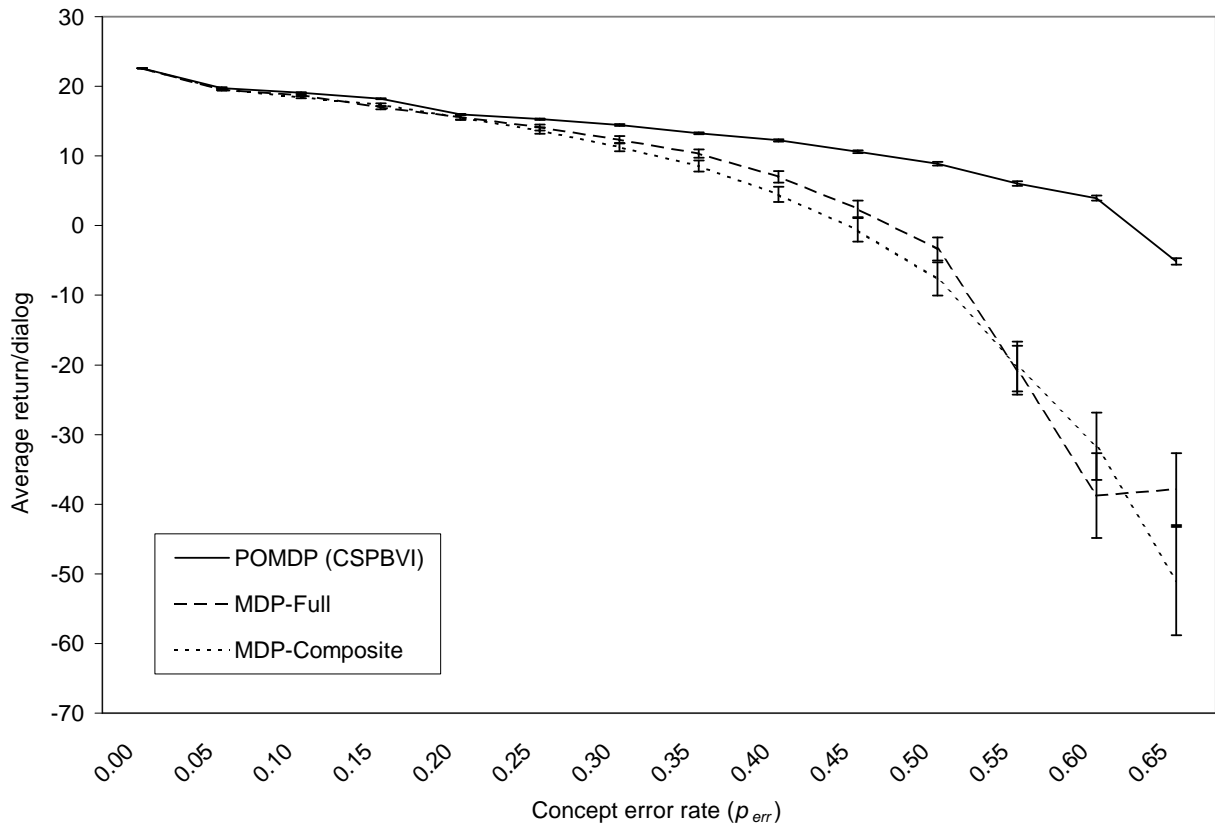


Figure 6.4 Concept error rate (p_{err}) vs. average return for the CSPBVI and two MDP baseline methods with no confidence score for MAXITRAVEL-W with 2 slots, each with 100 values).

and 6.8 shows the performance of the CSPBVI policies vs. MDP-2-Composite for various concept error rates ($p_{err} = \{0.1, 0.3, 0.5\}$). Each plot shows confidence score informativeness measures $h = \{0, 1, 2, 3, 4, 5\}$. On each, the X axis shows the number of slots, and the Y axis shows the average return gained *per slot*. Where recognition is not perfect (i.e., $p_{err} > 0$), CSPBVI outperforms the MDP-2 baseline, and as the confidence score becomes more informative (i.e., as h increases), performance at a given concept error rate p_{err} increases for both the POMDP and MDP policies, with greater increases observed for higher error rates.

Finally, to assess the overall benefits of automated planning, CSPBVI optimization was then compared to two handcrafted dialog managers. Confidence score information was not used (i.e., h was set to 0). Two controllers were formed: the first controller, called HC4-W, consisted of W instances of HC4 (Figure 5.9, page 83), one for each slot. At runtime, each slot's instance of HC4 nominated an action, and the same heuristic used by the CSPBVI controller (Algorithm 17) was then used to choose which action to execute among these. The second controller, called HC5-W, operated in the same way but was composed of 5 instances of HC5 (Figure 5.10, page 83).

HC4-W and HC5-W were evaluated by running 10,000 simulated dialogs for various number of slots and error rates. Results for various concept error rates ($p_{err} = 0.00, 0.10, 0.30, 0.50$) are shown in Figure 6.9. The X-axis shows the number of slots, and the Y-axis shows the average reward gained *per dialog per slot*, with error bars indicating the 95% confidence interval for the

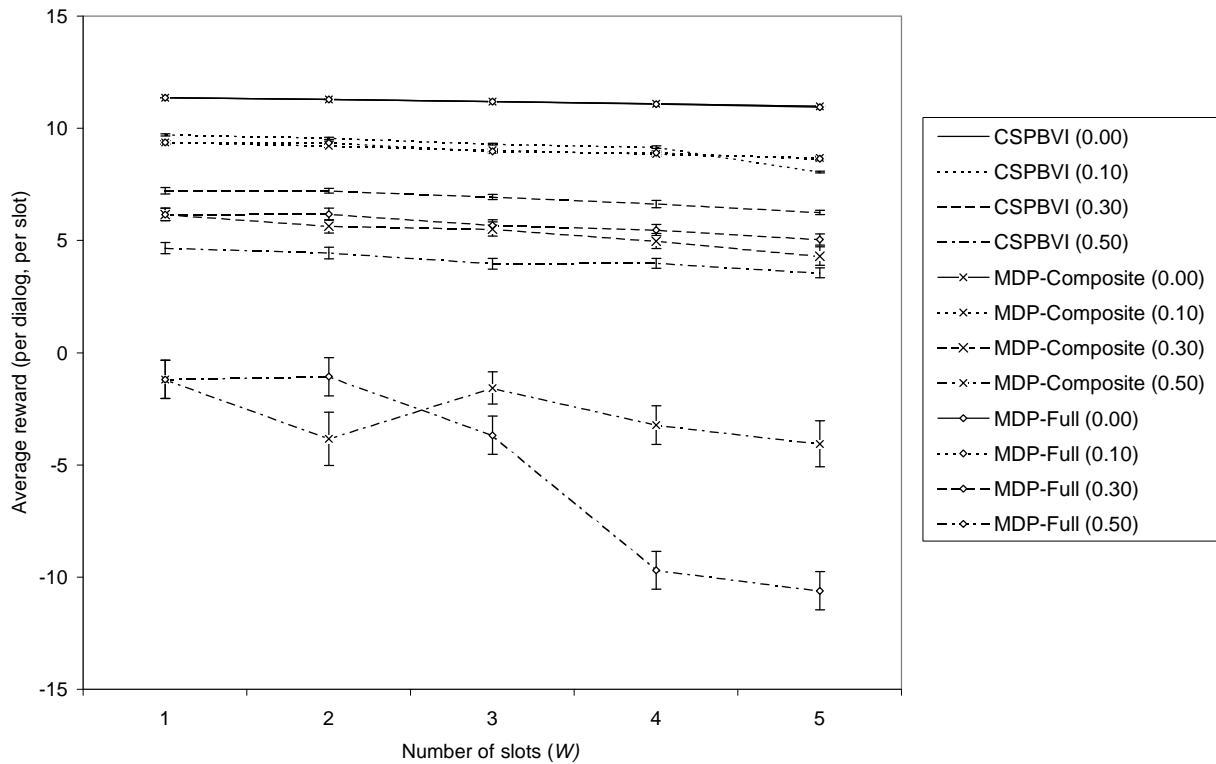


Figure 6.5 Number of slots vs. average return for CSPBVI and two MDP baselines for various concept error rates (values of p_{err}) for MAXITRAVEL-W.

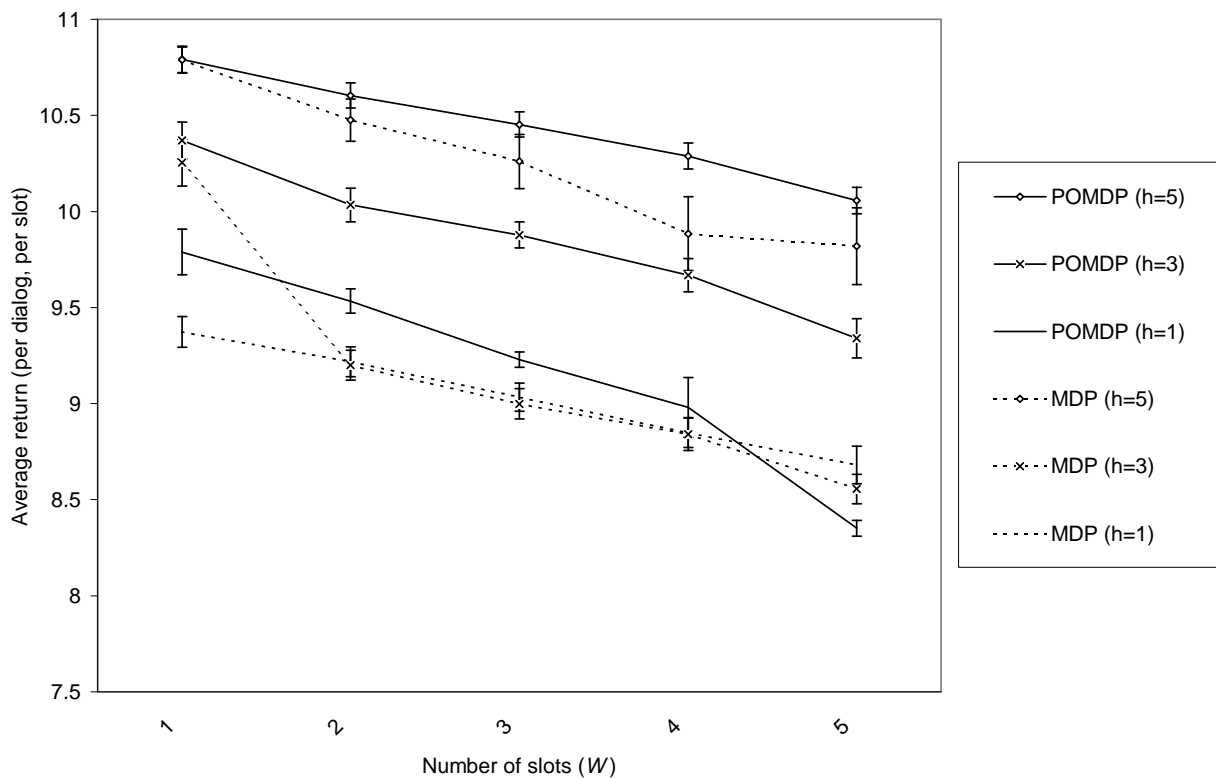


Figure 6.6 Number of slots vs. average return for CSPBVI and MDP-2 baseline for concept error rate $p_{err} = 0.10$ for MAXITRAVEL-W with varying levels of confidence score informativeness.

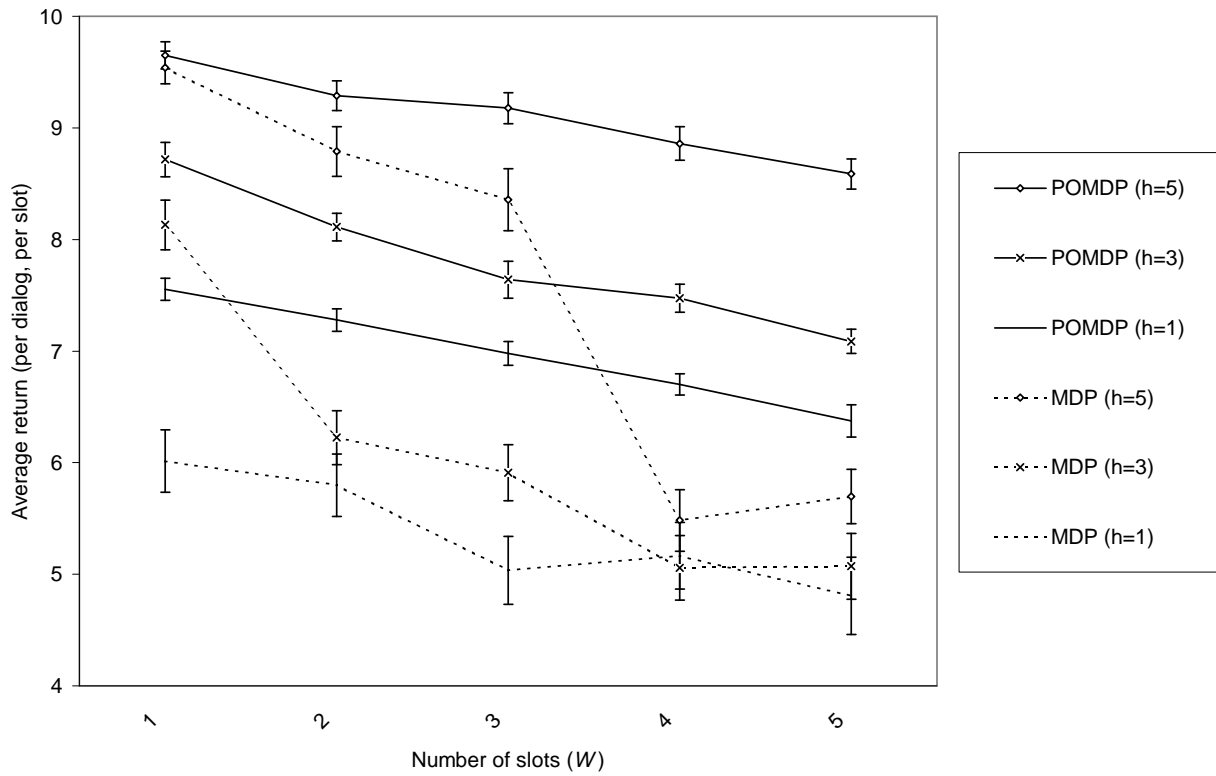


Figure 6.7 Number of slots vs. average return for CSPBVI and MDP-2 baseline for concept error rate $p_{err} = 0.30$ for MAXITRAVEL-W with varying levels of confidence score informativeness.

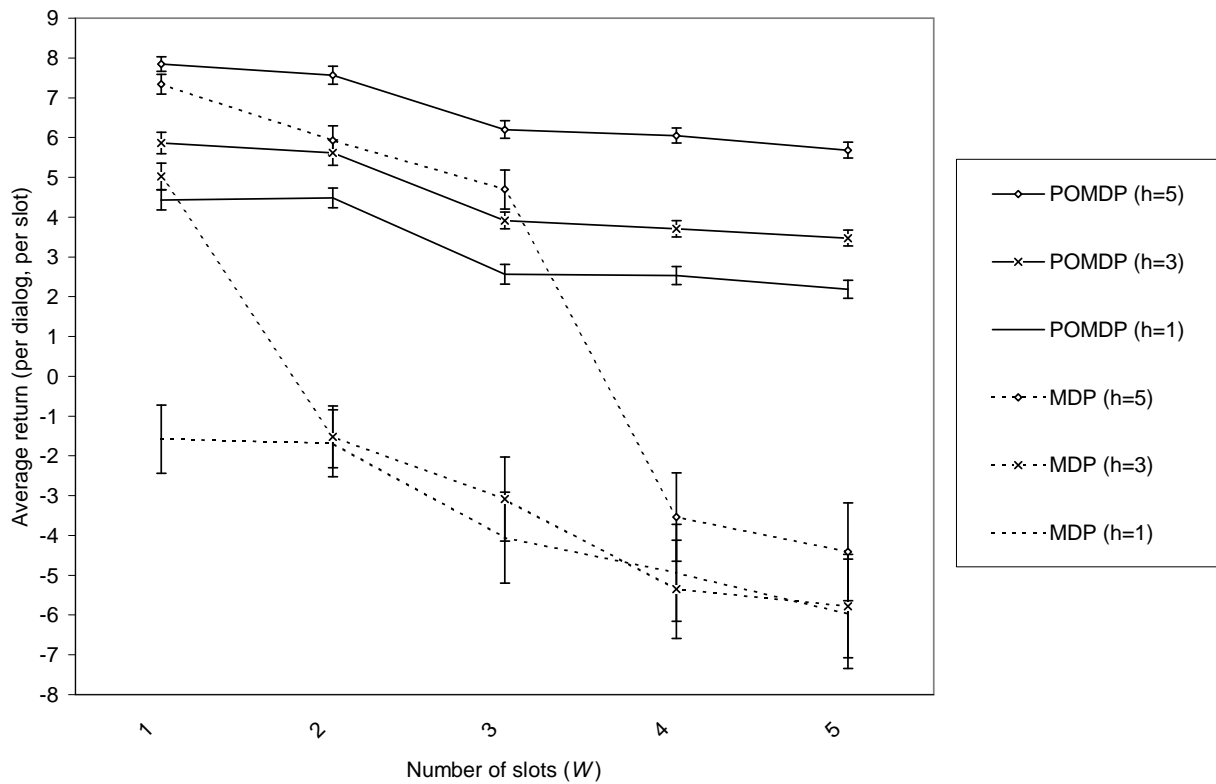


Figure 6.8 Number of slots vs. average return for CSPBVI and MDP-2 baseline for concept error rate $p_{err} = 0.50$ for MAXITRAVEL-W with varying levels of confidence score informativeness.

true average reward gained per dialog turn. The POMDP solution outperforms both handcrafted controllers at all error rates. As the number of slots increases, the reward gained per slot decreases, but at higher error rates (i.e., $p_{err} = 0.50$) this decline is precipitous for the handcrafted controllers but gradual for the POMDP, indicating the POMDP is more robust. One reason for this is that the POMDP is making use of a user model and taking proper account of observations. By contrast the handcrafted policies place equal trust in all observations. As dialogs become longer, the simulated user provides less-reliable information about *other* slots more times in each dialog, causing the performance of handcrafted policies to degrade.

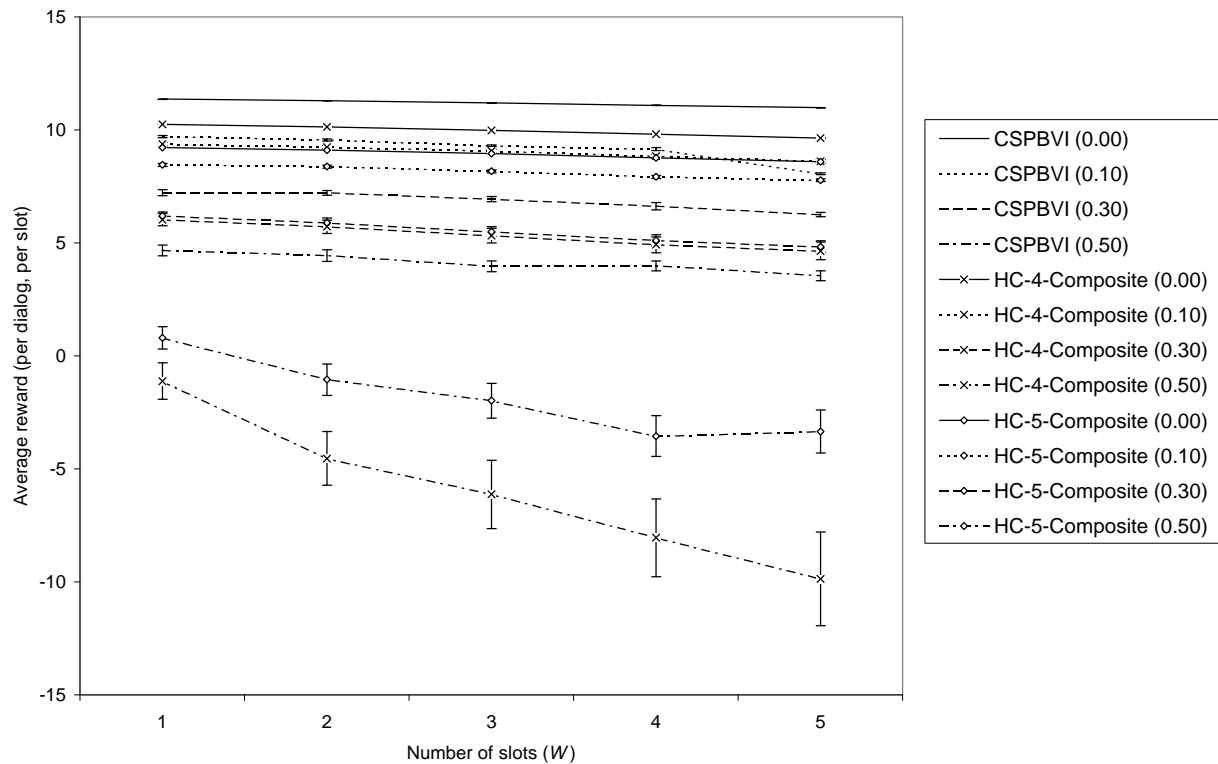


Figure 6.9 Number of slots vs. average return per dialog per slot for CSPBVI and the HC4-W and HC5-W handcrafted controller baselines for MAXITRAVEL-W with no confidence score information.

To this point, all experiments have been optimized and evaluated on the same user model (*training* user model in Table 6.2). In practice, a user model will likely be a noisy estimate of real user behavior, and experiments so far have not addressed what effect this deviation might have on performance. Thus a final experiment was conducted which creates a policy using the *training* user model and evaluates the policy using the *testing* user model.

A 5-slot MAXITRAVEL-W was created which used no confidence score information (i.e., $h = 0$). First, the “training” user model was installed into each slot, and CSPBVI optimization was performed using $N = 100$ belief point samples and $K = 50$ observation samples. The resulting policy was evaluated using the same (training) user model, by running 10,000 dialogs. Then, the “testing” user model was installed into each slot, and 10,000 dialogs were run with the policy and this user model. This was then repeated for concept error rates (p_{err}) ranging from 0.00 to

0.65, and results are shown in Figure 6.10. The Y-axis shows average reward gained per dialog *per slot*, and error bars indicate the 95% confidence interval for the true average reward gained per dialog turn. As speech recognition errors increase, the average reward per turn decreases as expected, and in general performance on the test user model is less than but very close to the training user model, implying that the method is reasonably robust to variations in patterns of user behavior or estimation errors in the user model parameters.

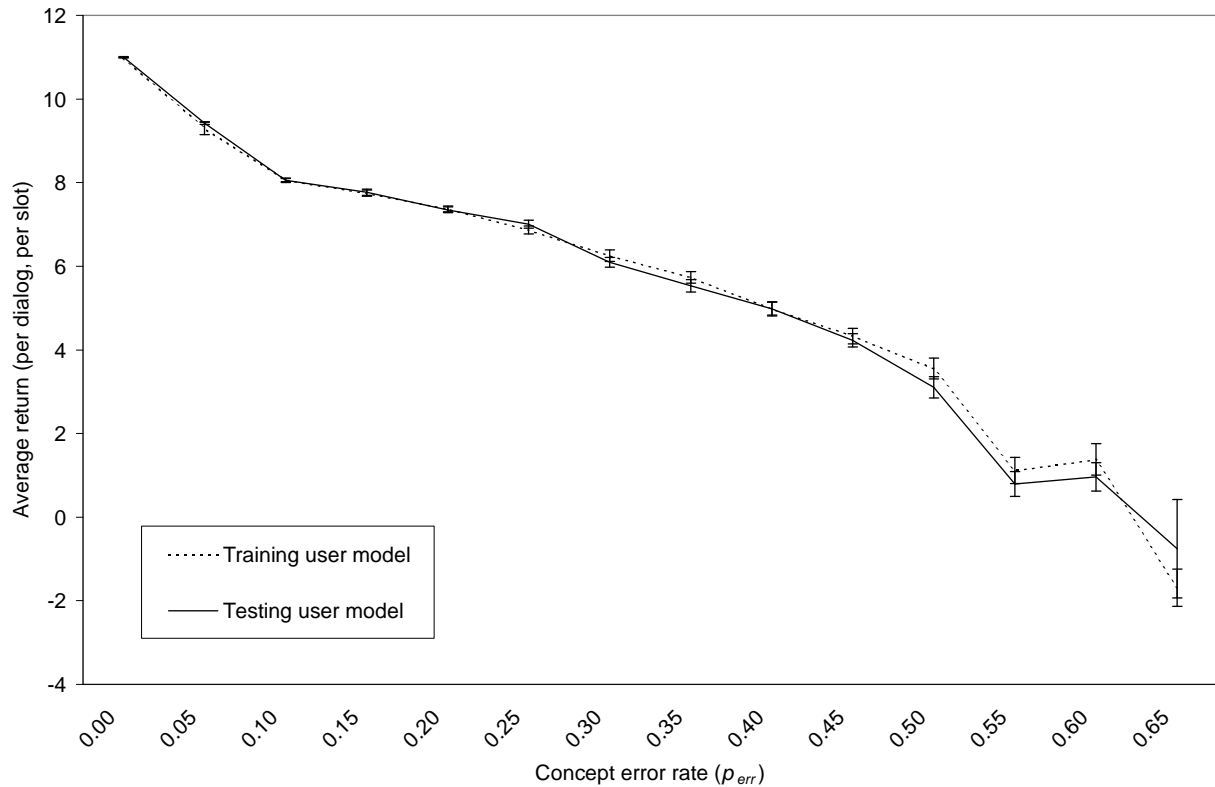


Figure 6.10 Concept error rate (p_{err}) vs. average return for the training and testing user models with 5 slots for MAXITRAVEL-W with no confidence score information.

In sum, CSPBVI provides a method to scale the SDS-POMDP model to slot-filling dialogs of a real-world size. Performance with respect to MDP and handcrafted baselines using simulated dialogs show that CSPBVI makes appropriate assumptions for the dialog domain, enabling POMDPs to scale while retaining the benefit of the POMDP formalism. Moreover, experiments with held-out dialog data show that CSPBVI policies appear to be robust to variations in user behavior, indicating that errors in user model estimation can be tolerated.

6.4 Application to a practical spoken dialog system

To verify that the CSPBVI method can function in real-world operating conditions as predicted in simulation, a complete slot-filling spoken dialog system was created called TOURISTTICKETS. TOURISTTICKETS consists of 4 slots which take between 3 and 11 values, shown in table 6.3. As in the examples above, the goal of the machine is to correctly identify the user's goal and submit

it – in this application, to print the appropriate ticket order. This task is based on a subset of the activities used for the SACTI data collection [125].

Slot	Cardinality	Values
<i>venue</i>	3	<i>castle, tower, museum</i>
<i>quantity</i>	10	1, 2, . . . , 10
<i>day</i>	7	<i>monday, tuesday, . . . , sunday</i>
<i>time</i>	9	9 AM, 10 AM, . . . , 5 PM

Table 6.3 Slot names and example values for the TOURISTTICKETS application.

TOURISTTICKETS was implemented in Microsoft Windows XP. For speech input, TOURISTTICKETS uses Cambridge University’s ATK real-time speech recognition engine with its standard acoustic models and confidence scoring mechanisms [129].⁴ A simple rule-based grammar was created which recognizes one or more slot values, yes and no, and carrier phrases such as “I’d like ... please”. For speech output, TOURISTTICKETS uses the *flite* text-to-speech engine with its standard models and settings [31].

A four-slot dialog manager was constructed as described in section 6.2, which uses the *training* user model estimated from the SACTI-1 corpus (Table 6.2, page 94). From experimentation with ATK, a concept error rate of $p_{err} = 0.35$ and a confidence score informativeness of $h = 3$ were estimated. CSPBVI optimization was performed in Matlab, and the belief monitoring and action selection algorithms were exported into a dynamically linked library using the Matlab compiler [46].

A simple phrase-spotting algorithm searches recognized utterances for slot-specific phrases such as “two tickets” or “two PM”, for phrases which are ambiguous between slots such as “two”, and also for variants of yes and no. At runtime, a basic ask/answer cycle is used in which the machine asks a question and waits for a response. If no recognition results is obtained after listening for 4 seconds, then the *null* observation is passed to the dialog manager. Otherwise, spotted phrases in the recognized string are entered into the POMDP as observations. After an observation is received, the dialog manager updates its belief state and selects the next action to take. The recognizer is run continuously, and the system implements barge-in so the user may speak at any time.

Figures 6.11 and 6.12 show screenshots of the system in operation. The foreground of Figure 6.11 shows the components of ATK. The small window labeled “aud” in the lower right is the audio input component, which samples incoming speech from the computer’s audio device. The energy of the speech is shown as a horizontal bar graph in this component, and the sampled speech is passed to the coder, the window on the central left labelled “code”. The central grayscale grid-like area of the coder shows a graphical representation of the feature vectors in time. The feature vectors are passed to the recognizer, the window in the upper left labelled “rec”. The results from the recognition are passed to the dialog manager, which is the large

⁴ATK is essentially a real-time interface to Cambridge University’s HTK speech recognition toolkit [45].

text window behind the ATK components (and shown on its own in Figure 6.12). There are two other ATK windows in Figure 6.11. First, the long window in the lower half of the screen labelled “aio” manages turn-taking: the dark sections on the top row show where the machine is speaking, and the light sections on the bottom row show where the machine heard the user speaking. Finally, the tall window on the right labelled “Monitor” is the monitor component and manages overall coordination of the components.

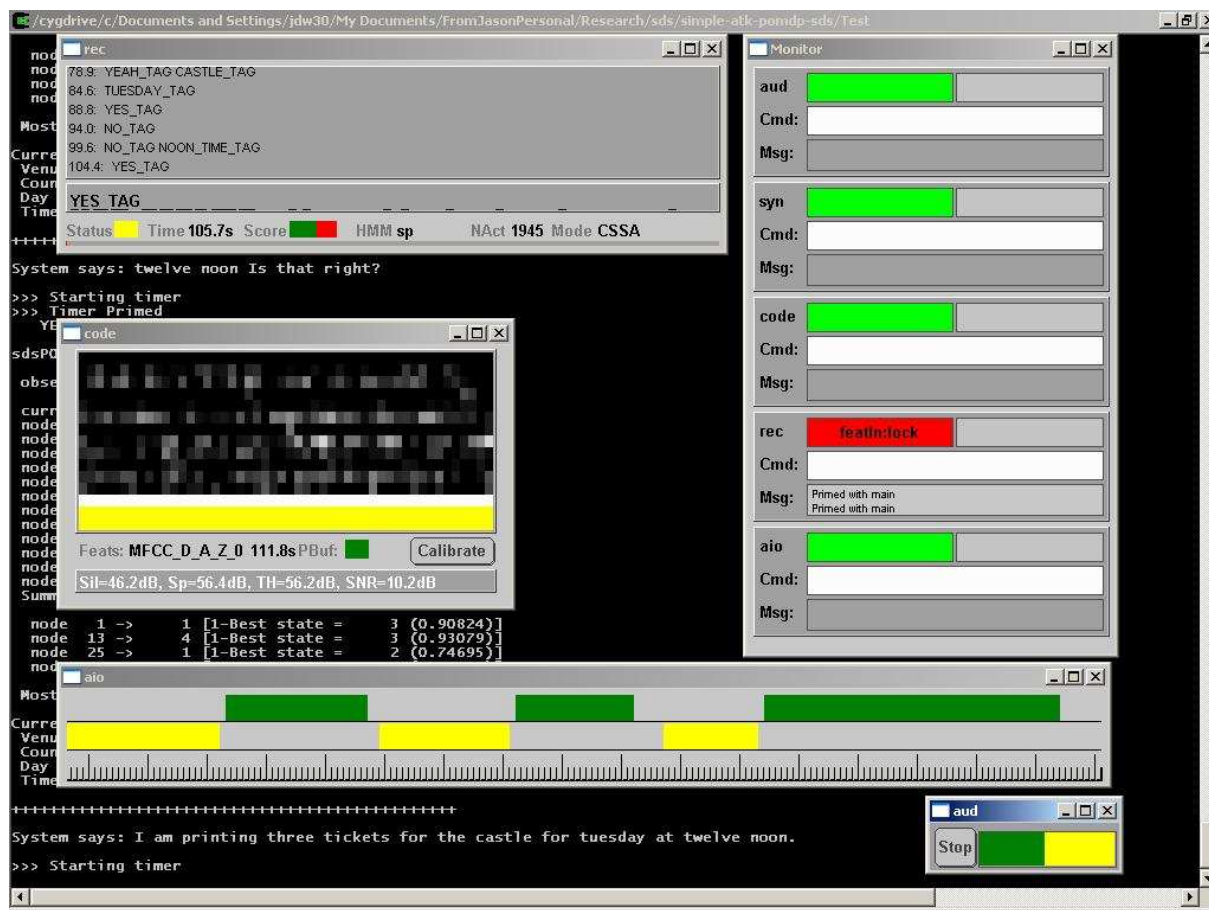


Figure 6.11 Screen shot of the TOURISTTICKETS application showing the ATK components in the foreground (see text for details).

Figure 6.12 shows the dialog manager component. The top portion of the screen shows turn n , and the very top of this portion shows $\hat{b}(best)$, the belief in the most likely user goal for each component. The four components shown are venue, ticket count, day, and time respectively. Just below this is s^* , the most likely user goal. In this turn, the most likely user goal for the *Time* slot is 8 AM, and the belief in this user goal is 0.1000.

The middle portion of the screen shows turn $n + 1$. The top of this section shows the system’s speech, and the second portion shows the speech recognized from the user in upper-case letters. Words ending in “TAG” are noted by the word-spotting algorithm. In turn $n + 1$ the user said “at noon” and this was recognized as “no noon”, from which the concepts $no()$ and $time(noon)$ are extracted and passed to the dialog manager as observations. The values in square brackets show

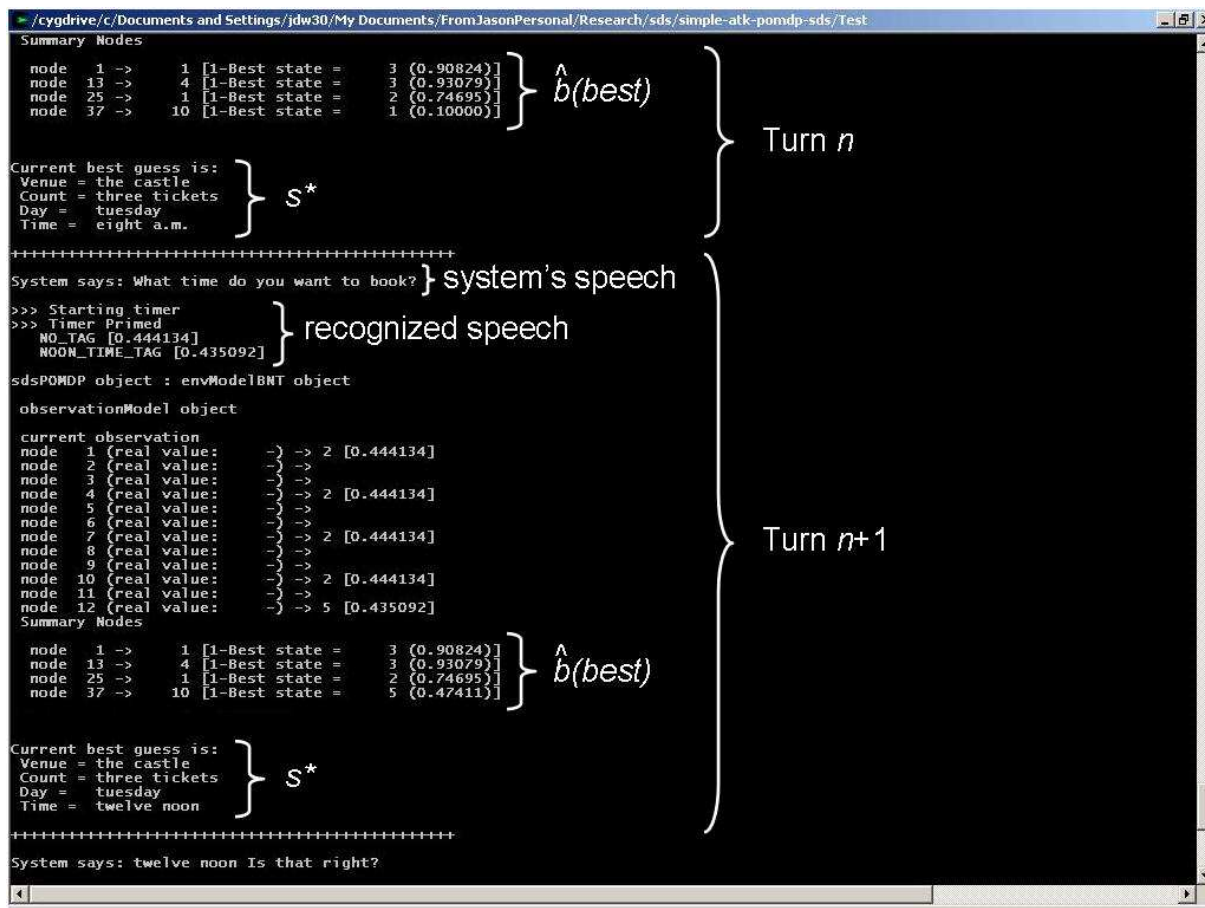


Figure 6.12 Screen shot of the TOURISTTICKETS application showing the dialog manager component. In turn n the most likely user goal for the time slot is 8 AM with belief 0.1000. In turn $n + 1$ the system asks “What time do you want to book?” and the user says “At noon” which is recognized as “no noon”. This evidence is entered into the dialog manager and results in the most likely user goal for the time slot changing to “twelve noon” with belief 0.47411.

the confidence scores for each of these words.⁵ After the observation has been entered, $\hat{b}(best)$ and s^* are updated, and these are shown in the lower part of turn $n + 1$. Note that now s^*_{time} has taken the value *noon* and now has mass $\hat{b}(\hat{s}_{time} = best) = 0.47411$. The next system action is to ask “Twelve noon, is that right?”.

In practice, the response time of the system is conducive to spoken conversation: once the end of the user’s speech has been detected, the system takes approximately 1 second to finish recognition, update its belief state, determine its next action, render the resulting action as speech (using the text-to-speech component), and begin playing it. Through using the system, the ordering of actions specified by the *chooseActionHeuristic* function (Algorithm 17, 95) was found to be more intuitive by changing it to first look for an *ask* then *confirm* action in the first slot, then an *ask* then *confirm* action in the second slot, and so on. The *submit* action is still

⁵The central area shows how the mechanics of how the observation is entered into the dialog manager, and isn’t important to this explanation.

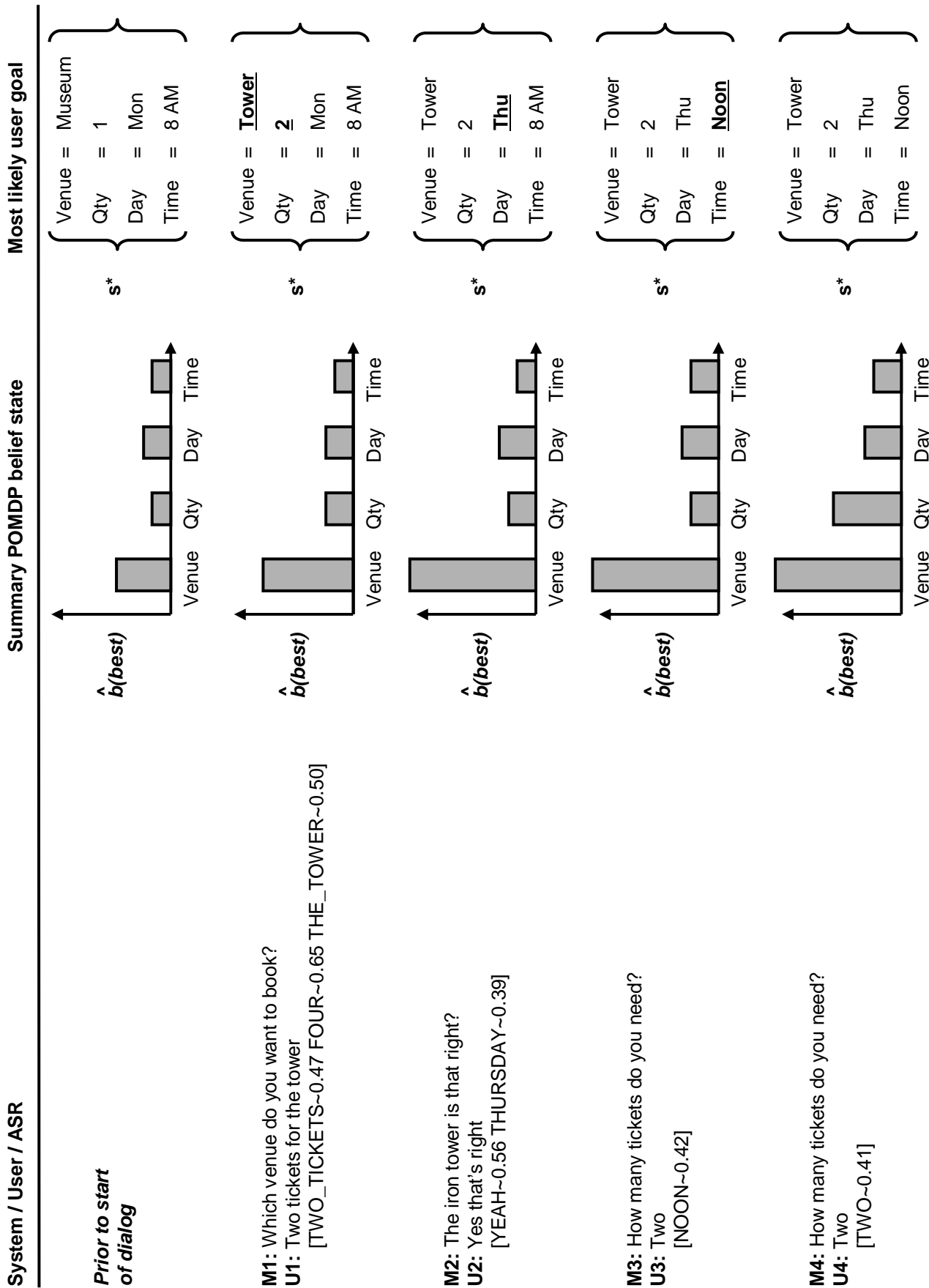


Figure 6.13 Sample conversation between a user and the TOURISTTICKETS spoken dialog system (first half, continued in Figure 6.14). The left column shows the machine's and user's speech, and the speech as it was recognized by the machine. The center column shows $\hat{b}(best)$ for each of the four slots. The right-hand column shows s^* , the most likely user goal at each time-step. Entries in underlined bold in the right-hand column have changed from the previous turn.

System / User / ASR	Summary POMDP belief state	Most likely user goal
<p>M5: Two tickets is that right? U5: Yes for Monday at noon [YEAH~0.63 SEVEN~0.18 MONDAY~0.37 NOON~0.48]</p>	<p>$\hat{b}(best)$</p>	<p>Venue = Tower Qty = 2 Day = <u>Thu</u> Time = Noon</p> <p>s^*</p>
<p>M6: Which day do you want to book? U6: Monday at noon [MONDAY~0.53 TEN~0.45 NOON~0.48]</p>	<p>$\hat{b}(best)$</p>	<p>Venue = Tower Qty = 2 Day = <u>Mon</u> Time = Noon</p> <p>s^*</p>
<p>M7: Monday is that right? U7: Yes at noon [YEAH~0.60 SEVEN~0.34 NOON~0.49]</p>	<p>$\hat{b}(best)$</p>	<p>Venue = Tower Qty = 2 Day = Mon Time = Noon</p> <p>s^*</p>
<p>M8: Twelve noon is that right? U8: Yes [YES~0.42]</p>	<p>$\hat{b}(best)$</p>	<p>Venue = Tower Qty = 2 Day = Mon Time = Noon</p> <p>s^*</p>
<p>M9: [prints two tickets for the tower for noon on Monday]</p>		

Figure 6.14 Sample conversation between a user and the TOURISTTICKETS spoken dialog system (second half, continued from Figure 6.13). The left column shows the machine's and user's speech, and the speech as it was recognized by the machine. The center column shows $\hat{b}(best)$ for each of the four slots. The right-hand column shows s^* , the most likely user goal at each time-step. Entries in underlined bold in the right-hand column have changed from the previous turn.

taken only once all slots nominate it.

A transcribed conversation with the system is shown in Figures 6.13 and 6.14. These figures show the words spoken by the machine and user, and the words recognized by the machine in the first column; the *belief* in the most likely user goal $\hat{b}(best)$ in the second column; and the *value* of the most likely user goal s^* in the third column. In this conversation the user is trying to buy two tickets for the tower at noon on Monday. At the start of the conversation (i.e., the first row of Figure 6.13) the distribution of belief over user goals is uniform – for example, the initial belief in each of the three possible *ticket venues* is 0.333 and in each of the ten possible *ticket quantities* is 0.1.

This conversation illustrates how the SDS-POMDP model copes with mis-recognitions in practice. For example, in U3, the user asks for “two” tickets but this is mis-recognized as “noon”. As a result, “noon” receives slightly more belief mass in the belief state, but the shift is minor because the user model predicts against this response. Even so, this dialog also demonstrates the value of accumulating these minor belief mass shifts: the user indicates the time of “noon” in U5, U6, and U7 even though it was never asked for. By turn M8, enough belief mass has been accumulated for the “noon” value that the machine confirms it rather than asking for it.

Finally, this dialog also shows how interpretation functions in practice. In U6 and U7 the user is asked about the “day” slot, and in both cases a number is mis-recognized in the user’s speech. In both cases this number is entered as an observation to the *quantity* and *time* slots but is ignored because the user model predicts that numbers *on their own* will never be responses to questions about days of the week. Had the mis-recognitions in U6 been “ten tickets” or “at ten” then a change in the belief state would have been observed. By contrast the (correct) recognition of “two” in U4 *does* cause a shift in belief mass since it is an expected response to a question about ticket quantity.

Overall, the TOURISTTICKETS application demonstrated that the SDS-POMDP model, optimized with the CSPBVI method, can be used to build a functional spoken dialog system, and that the behaviors predicted in simulation are realized in an end-to-end system.

Conclusions and future work

7.1 Thesis summary

Despite the advances made in recent years, the design of robust spoken dialog systems continues to be a major research challenge. A central problem is that the uncertainty caused by speech recognition errors leads to false assumptions about the dialog state, which in turn lead to spurious dialogs.

Various existing techniques have sought to improve robustness to speech recognition errors, but to date many of these have not been combined in a principled way. First, *confidence scores* quantify the reliability of a single speech recognition hypothesis, but taken in isolation it is unclear how to base decisions on a confidence score to serve a long-term dialog goal. *Automated planning* such as (fully-observable) Markov decision processes (MDPs) *do* choose actions to serve a long-term goal, yet these techniques approximate what is properly a distribution over all possible dialog states with a single hypothesis, an assumption which leads to a precipitous drop in performance as speech recognition errors become more prevalent. Conversely, other methods *have* successfully quantified uncertainty by maintaining *parallel hypotheses* for the current state of the dialog, yet these have not attempted to perform planning, and it is hard to see how these techniques will make long-term progress to a dialog goal. Finally, handcrafting a dialog manager is an iterative, expensive process which isn't guaranteed to achieve optimal results.

This thesis has argued that a partially observable Markov decision process (POMDP) provides a principled framework for modelling spoken man-machine dialog. Furthermore, by casting a dialog manager in the form of an SDS-POMDP, it has been shown that the ideas underlying existing techniques to improving robustness – multiple state hypotheses, confidence scores, and automated planning – are all special cases of the POMDP formalism. Thus, the POMDP approach provides a basis for both improving the performance of these existing techniques and unifying them into a single framework supporting global optimisation. Further, the SDS-POMDP model extends past work which has applied POMDPs to spoken dialog systems by constructing explicit, intuitive conditional probability tables which combine statistical models of user behavior and the speech recognition process with expert domain knowledge; by adding an element of dialog

history which enables dialog managers to account for the “appropriateness” of actions; and by naturally incorporating continuous recognition features such as a confidence score in the form of continuous observations.

This thesis then presented an example dialog application, called TRAVEL, and showed through dialog simulation that the theoretical advantages promised by the POMDP formalism are indeed borne out in practice. Moreover, the SDS-POMDP model was shown to outperform three traditional handcrafted dialog managers. However, the TRAVEL application also highlighted the key challenge of scalability: TRAVEL is far too small to be of practical use, and scaling it to a realistic size quickly defeats contemporary POMDP optimization techniques such as point-based value iteration (PBVI). The core problem of scalability was then tackled by limiting the scope of dialog applications to the so-called *slot-filling* domain, and by introducing a series of novel optimization techniques tailored to slot-filling dialogs. Summary point-based value iteration (SPBVI) addressed the problem of how to grow the number of values that one slot can take on – for example, the number of airports in a flight information application or the number of cities in a travel application. SPBVI simplifies optimization by restricting machine actions to operate on the user goal with the most belief mass, reducing the complexity of the planning problem to a small constant. Although SPBVI makes significant approximations, empirical results confirm that SPBVI outperforms baseline MDP and handcrafted methods while scaling to problems beyond the reach of current techniques.

SPBVI effectively reduces the dimensionality of the planning simplex from N^W to c^W , where N is the number of values per slot, W is the number of slots, and c is a small constant. For one slot ($W = 1$), the reduction in complexity is significant, but the complexity of SPBVI is still exponential in the number of slots W , and an extension of SPBVI called Composite SPBVI (CSPBVI) was introduced to scale to many slots. CSPBVI decomposes the overall planning problem into smaller planning problems by constructing policies separately for *each slot*. CSPBVI first samples the system dynamics of the entire application *locally* from the perspective of a single slot. Optimization is then performed using the dynamics sampled for that slot, producing one policy for each slot. At runtime, the policy for each slot nominates an action which it deems optimal for its slot, and a handcrafted heuristic chooses which of these nominations to take. CSPBVI also makes important theoretical assumptions which could compromise performance, but dialog simulation experiments show that CSPBVI scales the SDS-POMDP model to slot-filling problems of real-world size, outperforming baseline MDP and handcrafted techniques by significant margins. Finally, a real spoken dialog system called TOURISTTICKETS was described, verifying that the SDS-POMDP model, optimized with CSPBVI, can indeed be used to build real, functional spoken dialog systems.

7.2 Future work

The results from this thesis suggest several avenues of interesting research. A useful incremental improvement would be to extend the optimization techniques to consider the belief mass of the

top *N*-best hypotheses, rather than only the mass of the *single best* hypothesis. This extension would enable the machine to take actions such as “Was that *Boston* or *Austin*?”. A second interesting improvement to CSPBVI would be to replace the hand-crafted action selection heuristic with a search for an optimal function. For dialog systems with small numbers of actions, this search should be straightforward.

More broadly, the choice of appropriate reward functions and their relationship to established metrics of user performance such as the PARADISE scheme remain to be resolved [117, 116, 118]. It has been noted in other reinforcement learning work in the dialog domain that even small changes to the reward function can yield significant policy changes [62, 83], underscoring the need to use well-understood reward measures. It has been found that, across several domains, broadly the same factors in the same ratios contribute to user satisfaction [52], so it seems plausible that a reliable metric of user satisfaction could be incorporated into the reward measure. It would be interesting to examine transcripts from existing dialog systems to see what can be inferred about the reward function implicitly used by (human) dialog designers. In some contexts such as call centers, an obvious choice of reward measure is hard currency: real monetary costs can be associated with individual actions such as database accesses or phone line usage and with long-term outcomes such as an operator transfers, a customer hanging up a phone in disgust (and possibly discontinuing business with this establishment), or a successfully completed task (resulting in a continued business relationship).

There is also the related question of how models of user behaviour should be created and evaluated. Significant work has been done on creating user models for spoken dialog systems [29, 97, 98, 99, 96, 83]. In theory any probabilistic model of user behavior may be incorporated into the SDS-POMDP framework, and one of the strengths of the POMDP approach is that it naturally captures models of user behavior that contain hidden state. Even so, there is a lack of an accepted methodology for validating the accuracy of a user model in the spoken dialog field, and it has been confirmed quantitatively that dialog managers produced with a poor user model perform poorly when properly scrutinized [94, 95].

A useful extension to the user model would be a hidden component for user *type*, such as “expert” and “novice”. Over time, user models could be refined through interaction, and in the limit they could model individual users. Moreover, on-line adaptation of POMDPs – in which the model parameters and policy are *simultaneously* re-estimated – is a promising approach for POMDP-based dialog managers, as the user’s behaviors may change as the dialog policy changes. POMDP adaptation techniques are in early stages of development but some progress has been reported [47]. On-line learning in (fully-observable) MDPs has been well studied [28], and extensions or approximations for learning in POMDPs may be possible.

In slot-filling dialogs, the state of the user is easy to define, but in more complex genres such as negotiation or tutoring the notion of user state is rather more thorny. A host of annotation schemes for dialog exist [58, 119, 114, 73], and established measurements such as *Kappa* show whether an annotation scheme can be reliably applied [15]. However, a reliable annotation scheme is only a *necessary* condition: a useful annotation methodology should ultimately result

in increased average return, suggesting the need for a more sophisticated annotation assessment metric.

In addition, users' behavior, including prosody [104, 86], vocabulary [63] and phraseology [68, 103, 127, 124] can be influenced by small changes in the wording of system prompts, and modelling these accurately is an open research area. The user's environment (e.g., quiet office vs. noisy street corner) also has a significant effect on choice of optimal policy [90], and this could also be built into the user and speech recognition models. Finally, user data is expensive to collect, and difficult issues relating to data sparsity arise quickly.

Although the SDS-POMDP framework is a general approach for spoken dialog system design, the optimization techniques in this thesis have been limited to the slot-filling domain. Broadening the application of POMDPs to more diverse, complex dialog genres such as negotiation, tutoring, command and control, or entertainment presents additional challenges. First, more complex representations of user goal, user action, system action, and dialog history will be required. Whereas belief monitoring techniques presented in this thesis relied on enumerating all possible user goals, this becomes impossible when representing more intricate, hierarchical notions of dialog expressed by, for example, the "Information State Update" framework [59]. To deal with these very large (or infinite) state spaces, a promising method is to partition the space of user goals into a hierarchy of equivalence classes or partitions. Belief updating is then performed on partitions rather than states. By successively splitting partitions, the system can use the incoming evidence to gradually focus-in on the underlying states of interest without having to needlessly consider large numbers of low probability states. Recent work in the Artificial Intelligence community has begun to address the general problem of probabilistic inference with tree-structured variables [35, 102], and one specific implementation of this idea in the spoken dialog system domain is the Hidden Information State (HIS) dialog model [131]. The HIS model uses probabilistic context-free rules to describe the partition hierarchy. In effect, these rules form an ontology of the application domain and they enable user goals to be expressed in a top-down manner which directly reflects the order in which sub-topics are typically visited in conversation.

It is possible that other spoken dialog domains will find new benefits from the POMDP approach. This thesis has argued that the primary benefit to using POMDPs for spoken dialog management is to address the uncertainty introduced by the speech recognition channel, but in some domains, POMDPs may helpfully address other issues: for example, in the tutoring domain, the "state" of the student-user (i.e., whether or not the student understands a concept) is not observable, and in a troubleshooting domain (in which a system is helping a person to fix a device such as a computer), the state of the device is not observable. In both these examples, even if speech recognition were perfect, important state information is still unobservable, implying that there is potential benefit to the POMDP approach beyond simply redressing errors in the ASR communication channel.

It would also be interesting to explore how a POMDP-based dialog manager could improve the performance of a speech recognizer or semantic decoder. The POMDP belief state could be used to adjust the prior probabilities in these components, or a POMDP action could choose

among language models or semantic interpretation schemes.

Looking further into the future, POMDP-based dialog management holds promise for improving commercial spoken dialog systems, but industrial development practices would need to be considered. In particular, commercial software projects usually assume well-defined test cases listing inputs and expected outputs. It is unclear how POMDP policies represented as value functions can be tested within this framework, and representing policies as finite state automata might be better suited to commercial exploitation.

Finally, more broadly, the SPBVI and CSPBVI optimization algorithms are suitable for a general class of POMDPs in which the certainty in a variable's value *rather than the value itself* is sufficient for planning. It would be interesting to see how these algorithms perform in other domains. Ultimately, the definitive test of a POMDP-based dialog system must be evaluation using real users, and the next step is clearly to begin user trials.

Bibliography

- [1] H Alshawi and S Douglas. Variant transduction: A method for rapid development of interactive spoken interfaces. In *Proc SIGdial Workshop on Discourse and Dialogue, Aalborg, Denmark*, 2001.
- [2] KJ Astrom. Optimal control of markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965.
- [3] H Aust, M Oerder, F Seide, and V Steinbiss. A spoken language inquiry system for automatic train timetable information. *Philips Journal of Research*, 49:399–418, 1995.
- [4] J Austin. *How to Do Things with Words*. Harvard University Press, 1962.
- [5] B Balantine. Principles of voice user interface design. In *Proc Audio-Visual Input-Output Society (AVIOS) San Jose, California*, 2003.
- [6] B Balantine, DP Morgan, and WS Meisel. *How to build a speech recognition application*. Enterprise Intergration Group, 1999.
- [7] N Bernsen, H Dybkjaer, and L Dybkjaer. *Designing Interactive Speech Systems*. Springer-Verlag, 1998.
- [8] D Bohus, P Carpenter, C Jin, D Wilson, R Zhang, and AI Rudnicky. Is this conversation on track? In *Proc Eurospeech, Aalborg, Denmark*, 2001.
- [9] D Bohus and AI Rudnicky. Modeling the cost of misunderstandings in the CMU Communicator dialogue system. In *Proc IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), Madonna di Campiglio, Italy*, 2001.
- [10] D Bohus and AI Rudnicky. Integrating multiple knowledge sources for utterance-level confidence annotation in the CMU Communicator spoken dialog system. Technical Report CS190, Carnegie Mellon University, 2002.

- [11] D Bohus and AI Rudnicky. Ravenclaw: Dialogue management using hierarchical task decomposition and an expectation agenda. In *Proc Eurospeech, Geneva, Switzerland*, 2003.
- [12] D Bohus and AI Rudnicky. A principled approach for rejection threshold optimization in spoken dialog systems. In *Proc Eurospeech, Lisbon, Portugal*, 2005.
- [13] D Bohus and AI Rudnicky. Sorry, I didn't catch that! An investigation of non-understanding errors and recovery strategies. In *Proc SIGdial Workshop on Discourse and Dialogue, Lisbon*, 2005.
- [14] C Boutilier and D Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proc Thirteenth National Conference on Artificial Intelligence, Portland, OR, USA*, pages 1168–1175, 1996.
- [15] J Carletta. Assessing agreement on classification tasks: The kappa statistic. *Computational Linguistics*, 22(2):249–254, 1996.
- [16] AR Cassandra. “solve-pomdp” software package. As of January, 2006.
- [17] AR Cassandra, LP Kaelbling, and ML Littman. Acting optimally in partially observable stochastic domains. In *Proc Conf on Artificial Intelligence, (AAAI), Seattle*, 1994.
- [18] J Chu-Carroll and R Carpenter. Vector-based natural language call routing. *Journal of Computational Linguistics*, 25(30):361–388, 1999.
- [19] HH Clark and SE Brennan. *Grounding in Communication*, chapter 7. APA, 1991.
- [20] MH Cohen, JP Giangola, and J Balough. *Voice User Interface Design*. Addison Wesley, 2004.
- [21] MG Core and JF Allen. Coding dialogues with the DAMSL annotation scheme. In David Traum, editor, *Working Notes: AAAI Fall Symposium on Communicative Action in Humans and Machines*, pages 28–35, Menlo Park, California, 1997. American Association for Artificial Intelligence.
- [22] M Denecke. Informational characterization of dialogue states. In *Proc Intl Conf on Spoken Language Processing (ICSLP), Beijing*, 2000.
- [23] M Denecke. Rapid prototyping for spoken dialogue systems. In *Proc Intl Conf on Computational Linguistics (COLING), Taipei*, 2002.
- [24] M Denecke, K Dohsaka, and M Nakano. Learning dialogue policies using state aggregation in reinforcement learning. In *Proc Intl Conf on Speech and Language Processing (ICSLP), Jeju, Korea*, 2004.
- [25] Y Deng, M Mahajan, and A Acero. Estimating speech recognition error rate without acoustic test data. In *Proc Eurospeech, Geneva, Switzerland*, 2003.

- [26] C Doran, J Aberdeen, L Damianos, and L Hirschman. Comparing several aspects of human-computer and human-human dialogues. In *Proc SIGdial Workshop on Discourse and Dialogue, Aalborg, Denmark*, 2001.
- [27] A Drake. *Observation of a Markov process through a noisy channel*. PhD thesis, Massachusetts Institute of Technology, 1962.
- [28] M Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts Amherst, 2002.
- [29] W Eckert, E Levin, and R Pieraccini. User modeling for spoken dialogue system evaluation. Technical Report TR 97.33.1, AT&T Bell Labs Research, 1997.
- [30] G Evermann and P Woodland. Posterior probability decoding, confidence estimation and system combination. In *Proc Speech Transcription Workshop, Baltimore*, 2000.
- [31] FLITE text-to-speech engine. <http://www.speech.cs.cmu.edu/flite/>.
- [32] M Gabsdil and O Lemon. Combining acoustic and pragmatic features to predict recognition performance in spoken dialogue systems. In *Proc Association for Computational Linguistics (ACL), Barcelona*, 2004.
- [33] J Glass. Challenges for spoken dialogue systems. In *Proc IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), Colorado, USA*, 1999.
- [34] D Goddeau and J Pineau. Fast reinforcement learning of dialog strategies. In *Proc Intl Conf on Acoustics, Speech, and Signal Processing (ICASSP), Istanbul*, 2000.
- [35] E Gyftodimos and PA Flach. Hierarchical bayesian networks: A probabilistic reasoning model for structured domains. In *Proc Workshop on Development of Representations, International Conference on Machine Learning (ICML), Cambridge, MA, USA*, 2002.
- [36] EA Hansen. Solving POMDPs by searching in policy space. In *Proc Uncertainty in Artificial Intelligence (UAI), Madison, Wisconsin*, 1998.
- [37] EA Hansen and Z Feng. Dynamic programming for POMDPs using a factored state representation. In *Proc Fifth International Conference on AI Planning Systems, Breckenridge, CO, USA*, pages 130–139, 2000.
- [38] Y He and SJ Young. A data-driven spoken language understanding system. In *HLT/NAACL04 Workshop on Spoken Language Understanding for Conversational Systems, Boston, MA, USA*, 2003.
- [39] J Henderson, O Lemon, and K Georgila. Hybrid reinforcement/supervised learning for dialogue policies from Communicator data. In *Proc Workshop on Knowledge and Reasoning in Practical Dialog Systems, Intl Joint Conf on Artificial Intelligence (IJCAI), Edinburgh*, 2005.

- [40] H Higashinaka, M Nakano, and K Aikawa. Corpus-based discourse understanding in spoken dialogue systems. In *Proc Association for Computational Linguistics (ACL), Sapporo, Japan*, 2003.
- [41] J Hirschberg, DJ Litman, and M Swerts. Detecting misrecognitions and corrections in spoken dialogue systems from ‘aware’ sites. In *Proc Prosody in Speech Recognition and Understanding, New Jersey, USA*, 2001.
- [42] J Hirschberg, DJ Litman, and M Swerts. Identifying user corrections automatically in spoken dialogue systems. In *Proc Second Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL), Pittsburgh, PA, USA*, 2001.
- [43] J Hoey and P Poupart. Solving POMDPs with continuous or large observation spaces. In *Proc Intl Joint Conf on Artificial Intelligence (IJCAI), Edinburgh, Scotland*, 2005.
- [44] E Horvitz and T Paek. DeepListener: Harnessing expected utility to guide clarification dialog in spoken language systems. In *Proc Intl Conf on Spoken Language Processing (ICSLP), Beijing*, 2000.
- [45] HTK speech recognition toolkit. <http://htk.eng.cam.ac.uk/>.
- [46] Mathworks Inc. MATLAB and MATLAB compiler. <http://www.mathworks.com/>.
- [47] R Jaulmes, J Pineau, and D Precup. Active learning in partially observable markov decision processes. In *European Conference on Machine Learning (ECML), Porto, Portugal*, 2005.
- [48] S Jekat, A Klein, E Maier, I Maleck, M Mast, and JJ Quantz. Dialogue acts in Verbmobil. Technical report Verbmobil Report 65, Universitat Hamburg, DFKI Saarbrucken, Universitat Erlangen, TU Berlin, 1995.
- [49] F Jensen. *Bayesian networks and decision graphs*. Springer Verlag, 2001.
- [50] D Jurafsky and J Martin. *Speech and language processing*. Prentice-Hall, 2000.
- [51] L Kaelbling, ML Littman, and AR Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.
- [52] CA Kamm, MA Walker, and DJ Litman. Evaluating spoken language systems. In *Proc American Voice Input/Output Society (AVIOS)*, 1999.
- [53] T Kemp and T Schaff. Estimating confidence using word lattices. In *Proc Eurospeech, Rhodes, Greece*, pages 827–830, 1997.
- [54] E Kraemer, M Swerts, M Theune, and M Weegels. Error detection in spoken human-machine interaction. *Intl Journal of Speech Technology*, 4(1):19–30, 1999.

- [55] E Krahmer, M Swerts, M Theune, and M Weegels. Problem spotting in human-machine interaction. In *Proc Eurospeech, Budapest*, pages 1423–1426, 1999.
- [56] IR Lane, S Ueno, and T Kawahara. Cooperative dialogue planning with user and situation models via example-based training. In *Proc Workshop on Man-Machine Symbiotic Systems, Kyoto, Japan*, pages 2837–2840, 2004.
- [57] I Langkilde, MA Walker, J Wright, A Gorin, and DJ Litman. Automatic prediction of problematic human-computer dialogues in “How may I help you?”. In *Proc IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), Colorado, USA*, pages 369–372, 1999.
- [58] S Larsson. Coding schemas for dialogue moves. Technical report, Department of Linguistics, Goteborg University, 1998.
- [59] S Larsson and D Traum. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering*, 5(3/4):323–340, 2000.
- [60] E Levin and R Pieraccini. A stochastic model of computer-human interaction for learning dialog strategies. In *Proc Eurospeech, Rhodes, Greece*, 1997.
- [61] E Levin, R Pieraccini, and W Eckert. Using Markov decision process for learning dialogue strategies. In *Proc Intl Conf on Acoustics, Speech, and Signal Processing (ICASSP), Seattle*, pages 201–204, 1998.
- [62] E Levin, R Pieraccini, and W Eckert. A stochastic model of human-machine interaction for learning dialogue strategies. *IEEE Trans on Speech and Audio Processing*, 8(1):11–23, 2000.
- [63] GA Levoa. Learning to speak to a spoken language system: Vocabulary convergence in novice users. In *Proc SIGdial Workshop on Discourse and Dialogue, Sapporo, Japan*, 2003.
- [64] GA Levow. Characterizing and recognizing spoken corrections in human-computer dialogue. In *Proc Association for Computational Linguistics and Intl Conf on Computational Linguistics (ACL/COLING), Montreal*, 1998.
- [65] DJ Litman, J Hirschberg, and M Swerts. Predicting user reactions to system error. In *Proc Association for Computational Linguistics (ACL), Toulouse, France*, 2001.
- [66] DJ Litman and S Pan. Predicting and adapting to poor speech recognition in a spoken dialogue system. In *Proc Conf on Artificial Intelligence (AAAI), Austin, Texas, USA*, pages 722–728, 2000.
- [67] S McGlashan, DC 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. Technical specification, W3C, 2004.

- [68] F McInnes, IA Nairn, DJ Attwater, and MA Jack. Effects of prompt style on user responses to an automated banking service using word-spotting. *BT Technical Journal*, 17(1):160–171, 1999.
- [69] GE Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.
- [70] R Moore and S Browning. Results of an exercise to collect “genuine” spoken enquiries using WoZ techniques. *Proc Institute of Acoustics*, 14(6), 1992.
- [71] PJ Moreno, B Logan, and B Raj. A boosting approach for confidence scoring. In *Proc Eurospeech, Aalborg, Denmark*, 2001.
- [72] R De Mori. *Spoken Dialogues with Computers*. Academic Press, 1998.
- [73] C Muller and M Strube. Multi-level annotation in MMAX. In *Proc SIGdial Workshop on Discourse and Dialogue, Sapporo, Japan*, 2003.
- [74] K Murphy. A survey of POMDP solution techniques. Technical report, U. C. Berkeley, 2000.
- [75] HI Ng and KT Lua. Dialogue input ranking in a multi-domain environment using transferable belief model. In *Proc SIGdial Workshop on Discourse and Dialogue, Sapporo, Japan*, 2003.
- [76] T Paek and E Horvitz. Uncertainty, utility, and misunderstanding: A decision-theoretic perspective on grounding in conversational systems. In *AAAI Fall Symposium on Psychological Models of Communication, North Falmouth, MA, USA*, pages 85–92, 1999.
- [77] T Paek and E Horvitz. Conversation as action under uncertainty. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI), Stanford, CA*, 2000.
- [78] T Paek and E Horvitz. Grounding criterion: Toward a formal theory of grounding. Technical Report MSR-TR-2000-40, Microsoft Research, 2000.
- [79] T Paek, E Horvitz, and E Ringger. Continuous listening for unconstrained spoken dialog. In *Proc Intl Conf on Spoken Language Processing (ICSLP), Beijing*, 2000.
- [80] C Pao, P Schmid, and J Glass. Confidence scoring for speech understanding systems. In *Proc Intl Conf on Speech and Language Processing (ICSLP), Sydney*, 1998.
- [81] R Pieraccini, E Levin, and W Eckert. Amica: the AT&T mixed initiative conversational architecture. In *Proc Eurospeech, Rhodes, Greece*, pages 1875–1878, 1997.
- [82] R Pieracinni and J Huerta. Where do we go from here? Research and commercial spoken dialog systems (invited talk). In *Proc SIGdial Workshop on Discourse and Dialogue, Lisbon*, 2005.

- [83] O Pietquin. *A framework for unsupervised learning of dialogue strategies*. PhD thesis, Faculty of Engineering, Mons (TCTS Lab), Belgium, 2004.
- [84] O Pietquin and S Renals. ASR modelling for automatic evaluation and optimisation of dialogue systems. In *Proc Intl Conf on Acoustics, Speech, and Signal Processing (ICASSP), Florida, USA, 2002*.
- [85] J Pineau, G Gordon, and S Thrun. Point-based value iteration: an anytime algorithm for POMDPs. In *Proc Intl Joint Conf on Artificial Intelligence (IJCAI), Acapulco, Mexico, 2003*.
- [86] H Pirker and G Loderer. “I said TWO TI-CKETS”: How to talk to a deaf wizard. In *Proc ESCA Workshop on Dialogue and Prosody, Veldhoven, The Netherlands, 1999*.
- [87] P Poupart and C Boutilier. VDCBPI: an approximate scalable algorithm for large scale POMDPs. In *Proc Advances in Neural Information Processing Systems 17 (NIPS), Vancouver, Canada, pages 1081–1088, 2004*.
- [88] ML Puterman. *Markov Decision Processes*. John Wiley and Sons, 1994.
- [89] LR Rabiner. *A tutorial on hidden Markov models and selected applications in speech recognition*, pages 267–296. Morgan Kaufmann Publishers, Inc., 1990.
- [90] A Raux, B Langner, D Bohus, A Black, and M Eskenazi. Let’s go public! Taking a spoken dialog system to the real world. In *Proc Eurospeech, Lisbon, Portugal, 2005*.
- [91] N Roy and G Gordon. Exponential family PCA for belief compression in POMDPs. In *Proc Advances in Neural Information Processing Systems (NIPS), Vancouver, BC, Canada, pages 1635–1642, 2002*.
- [92] N Roy, J Pineau, and S Thrun. Spoken dialog management for robots. In *Proc Association for Computational Linguistics (ACL), Hong Kong, 2000*.
- [93] D Sadek, P Bretier, V Cadoret, A Cozannet, P Dupont, P Ferrieux, and F Panaget. A cooperative spoken dialogue system based on a rational agent model: A RST implementation of the AGS application. In *Proc ESCA/ETR Workshop on Spoken Dialogue Systems, Vigs, Denmark, pages 145–148, 1995*.
- [94] J Schatzmann, K Georgila, and SJ Young. Quantitative evaluation of user simulation techniques for spoken dialogue systems. In *Proc SIGdial Workshop on Discourse and Dialogue, Lisbon, 2005*.
- [95] J Schatzmann, MN Stuttle, K Weilhammer, and SJ Young. Effects of the user model on simulation-based learning of dialogue strategies. In *Proc IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), San Juan, Puerto Rico, USA, 2005*.
- [96] K Scheffler. *Automatic design of spoken dialogue systems*. PhD thesis, Cambridge University, 2002.

- [97] K Scheffler and SJ Young. Probabilistic simulation of human-machine dialogues. In *Proc Intl Conf on Acoustics, Speech, and Signal Processing (ICASSP), Istanbul, 2000*.
- [98] K Scheffler and SJ Young. Corpus-based dialogue simulation for automatic strategy learning and evaluation. In *Proc North American Association for Computational Linguistics (NAACL) Workshop on Adaptation in Dialogue Systems, Pittsburgh, USA, 2001*.
- [99] K Scheffler and SJ Young. Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. In *Proc Human Language Technologies (HLT), San Diego, USA, 2002*.
- [100] J Searle. A classification of illocutionary acts. *Language in Society*, 5:1–23, 1976.
- [101] S Seneff and J Polifroni. Dialogue management in the Mercury flight reservation system. In *Proc Satellite Dialogue Workshop, Conference of the North American Chapter of the Association for Computational Linguistics and Conference on Applied Natural Language Processing (ANLP-NAACL), Seattle, WA, 2000*.
- [102] R Sharma and D Poole. Probabilistic reasoning with hierarchically structured variables. In *Proc Intl Joint Conf on Artificial Intelligence (IJCAI), Edinburgh, 2005*.
- [103] T Sheeder and J Balough. Say it like you mean it: Priming for structure in caller responses to a spoken dialogue system. *International Journal of Speech Technology*, 6:103–111, 2003.
- [104] J Shin, S Narayanan, L Gerber, A Kazemzadeh, and D Byrd. Analysis of user behavior under error conditions in spoken dialogue. In *Proc Intl Conf on Spoken Language Processing (ICSLP), Colorado, USA, 2002*.
- [105] S Singh, DJ Litman, M Kearns, and MA Walker. Optimizing dialogue management with reinforcement learning: experiments with the NJFun system. *Journal of Artificial Intelligence*, 16:105–133, 2002.
- [106] G Skantze. Exploring human error handling strategies: Implications for spoken dialogue systems. In *Proc ISCA Tutorial and Research Workshop on Error Handling in Spoken Dialogue Systems, Château d'Oex, Vaud, Switzerland, 2003*.
- [107] RD Smallwood and EJ Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.
- [108] R Smith and D Hipp. *Spoken Natural Language Dialog Systems: A Practical Approach*. Oxford University Press, 1994.
- [109] EJ Sondik. *The optimal control of partially observable Markov decision processes*. PhD thesis, Stanford University, 1971.

- [110] EJ Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978.
- [111] MTJ Spaan and N Vlassis. Perseus: randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.
- [112] MN Stuttle, JD Williams, and SJ Young. A framework for Wizard-of-Oz experiments with a simulated ASR channel. In *Proc Intl Conf on Speech and Language Processing (ICSLP)*, Jeju, Korea, 2004.
- [113] D Traum. *A Computational Theory of Grounding in Natural Language Conversation*. PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, USA, 1994.
- [114] D Traum. 20 questions for dialogue act taxonomies. *Journal of Semantics*, 17:7–30, 2000.
- [115] MA Walker, JC Fromer, and S Narayanan. Learning optimal dialogue strategies: a case study of a spoken dialogue agent for email. In *Proc Association for Computational Linguistics and Intl Conf on Computational Linguistics (ACL/COLING)*, Montreal, 1998.
- [116] MA Walker, CA Kamm, and DJ Litman. Towards developing general models of usability with PARADISE. *Natural Language Engineering: Special Issue on Best Practice in Spoken Dialogue Systems*, 2000.
- [117] MA Walker, DJ Litman, CA Kamm, and A Abella. PARADISE: A framework for evaluating spoken dialogue agents. In *Proc Association for Computational Linguistics (ACL)*, Madrid, Spain, 1997.
- [118] MA Walker, DJ Litman, CA Kamm, and A Abella. Evaluating spoken dialogue agents with PARADISE: Two case studies. *Computer Speech and Language*, 1998.
- [119] MA Walker and R Passonneau. DATE: A dialogue act tagging scheme for evaluation of spoken dialogue systems. In *Proc Conf on Human Language Technology (HLT)*, San Diego, 2001.
- [120] MA Walker, AI Rudnicky, R Prasad, J Aberdeen, EO Bratt, J Garofolo, H Hastie, A Le, B Pellom, A Potamianos, R Passonneau, S Roukos, G Sanders, S Seneff, and D Stallard. DARPA Communicator: Cross-system results for the 2001 evaluation. In *Proc Intl Conf on Spoken Language Processing (ICSLP)*, Colorado, USA, 2002.
- [121] CJCH Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.
- [122] JD Williams, P Poupart, and SJ Young. Factored partially observable Markov decision processes for dialogue management. In *Proc Workshop on Knowledge and Reasoning in Practical Dialog Systems, Intl Joint Conf on Artificial Intelligence (IJCAI)*, Edinburgh, 2005.

- [123] JD Williams, P Poupart, and SJ Young. Partially observable Markov decision processes with continuous observations for dialogue management. In *Proc SIGdial Workshop on Discourse and Dialogue, Lisbon, 2005*.
- [124] JD Williams, AT Shaw, LJ Piano, and M Abt. Preference, perception, and task completion of open, menu-based, and directed prompts for call routing: a case study. In *Proc Eurospeech, Geneva, Switzerland, 2003*.
- [125] JD Williams and SJ Young. Characterizing task-oriented dialog using a simulated ASR channel. In *Proc Intl Conf on Speech and Language Processing (ICSLP), Jeju, Korea, 2004*.
- [126] JD Williams and SJ Young. Scaling up POMDPs for dialog management: The “summary POMDP” method. In *Proc IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), San Juan, Puerto Rico, USA, 2005*.
- [127] SM Witt and JD Williams. Two studies of open vs. directed dialog strategies in spoken dialog systems. In *Proc Eurospeech, Geneva, Switzerland, 2003*.
- [128] SJ Young. Probabilistic methods in spoken dialogue systems. *Philosophical Trans of the Royal Society (Series A)*, 1769(358):1389–1402, 2000.
- [129] SJ Young. ATK: An application toolkit for HTK. Technical report, Cambridge University Engineering Department, 2004.
- [130] SJ Young and C Proctor. The design and implementation of dialogue control in voice operated database inquiry systems. *Computer Speech and Language*, 4(3):329–353, 1989.
- [131] SJ Young, JD Williams, J Schatzmann, MN Stuttle, and K Weilhammer. The hidden information state approach to dialogue management. Technical Report CUED/F-INFENG/TR.544, Cambridge University Engineering Department, 2006.
- [132] B Zhang, Q Cai, J Mao, E Chang, and B Guo. Spoken dialogue management as planning and acting under uncertainty. In *Proc Eurospeech, Aalborg, Denmark, 2001*.
- [133] B Zhang, Q Cai, J Mao, and B Guo. Planning and acting under uncertainty: A new model for spoken dialogue system. In *Proc Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 572–579, 2001.
- [134] V Zue. Conversational interfaces: Advances and challenges. In *Proc Eurospeech, Rhodes, Greece, 1997*.