Automatic Design of Spoken Dialogue Systems

Konrad Haarhoff Scheffler

St. John's College
and
Cambridge University Engineering Department

April 2002

Dissertation submitted to the University of Cambridge
for the degree of Doctor of Philosophy

“Speak when you’re spoken to!” the Red Queen sharply interrupted her.
“But if everybody obeyed that rule,” said Alice, . . .

Lewis Carrol, “Through the looking-glass and what Alice found there”

Abstract

The field of spoken dialogue systems has developed rapidly in recent years. However, optimisation, evaluation and rapid development of systems remain problematic. This dissertation addresses these issues by investigating ways to automate part of the design process.

The first contribution is the development of a simulation tool for approximate dialogue evaluation during the design process. Detailed probabilistic, data-driven models of user behaviour and recognition errors are developed. This differs from existing approaches, which are either not data-driven or lack the detail required to produce accurate results.

The models are trained using data from an early field trial and can be used to produce dialogues between a simulated user and the system under development. The simulations produce dialogues that correspond closely to real dialogues in a practical application.

Next, the simulation tool is applied in a framework for automatically designing and optimising dialogue management strategies. The dialogue is modelled as a Markov decision process and a reinforcement learning algorithm used to find an optimal strategy.

It is verified that this learning approach can find strategies which are adapted to different model parameters and different evaluation functions. Both of these factors will typically vary across different applications. When tested on a simulation of a practical application, the automatically learned strategies give a performance gain over standard handcrafted strategies. The results are also comparable to those obtained by a complete manual design which uses a much larger state space and has been through several iterations of refinement.

The dissertation concludes with a discussion and suggestions for future research.

Keywords: human-computer dialogue, human-computer interaction, dialogue system, spoken dialogue systems, dialogue simulation, dialogue system evaluation, dialogue management, dialogue strategy, dialogue optimisation, user modelling, strategy learning, Markov decision process, reinforcement learning, Q-learning.

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where stated. It has not been submitted in whole or part for a degree at any other university. The length of this dissertation including footnotes does not exceed 40000 words and it contains 27 figures.

Acknowledgements

I would like to thank my supervisor Steve Young for his expert guidance during the course of my studies. His advice has been invaluable.

I am grateful to the Association of Commonwealth Universities for sponsoring my studies, the British Council for administering my scholarship and St. John's College and the Cambridge University Engineering Department for additional funding.

Finally, thanks are due to everyone at CUED who contributed to the smooth handling of administrative and technical details, as well as my friends and colleagues for creating an incredibly stimulating environment. It has been a pleasure to work here.

Abbreviations

ASR	Automatic Speech Recognition
FSM	Finite State Machine
IDS	Interactive Dialogue System
ILP	Inductive Logic Programming
MDP	Markov Decision Process
NLU	Natural Language Understanding
POMDP	Partially Observable Markov Decision Process
SDP	Sequential Decision Process
SDS	Spoken Dialogue System

Contents

List of Figures

List of Tables

1

Introduction

1.1 Human-machine dialogue

The idea of a computer that can engage in natural, intelligent conversation with human beings has long been a sought-after ideal. It is natural to want to use spoken language as a means of interaction, even if what we are interacting with is an inanimate machine.

Recently, technology has finally started turning this ideal into reality. Although machines cannot yet converse about their innermost thoughts and feelings, it is now possible to dial a telephone number and, by simply speaking to it, ask a computer to reserve a cinema ticket, tell you what the weather is going to be like, or read back any email you may have received. So what we mean by a dialogue between a human and a machine is literally a spoken conversation, with both participants listening, understanding and reacting to what the other is saying.

More generally, a human-machine dialogue can also be conducted using means other than speech. For instance, research is being done to support human-machine communication by means of gestures, and visual output on a computer screen is a popular means of conveying information. In fact, any interaction based on sequential turn taking between two agents is an example of a dialogue, irrespective of whether the method of communication between the agents is speech, gestures, touch screens, visual displays, or a combination thereof. Computer systems designed to interact with humans in this way are called interactive dialogue systems (IDS), and systems for which the means of communication is specifically speech are called spoken dialogue systems (SDS).

Unfortunately, the design of dialogue systems is by no means easy: for each new dialogue application that is developed, many difficult decisions have to be made by design experts. One such class of decisions involves questions on how the system should react to the dialogue situation in which it finds itself, in order to maximise the efficiency and accuracy of the information exchange. For instance, how should the system go about confirming what it believes the human user has said without unduly disrupting the flow of the dialogue? How should it manage the conversational initiative it allows the user in order to make the interaction flow as naturally as possible? It is questions such as these that motivate the research presented in this thesis.

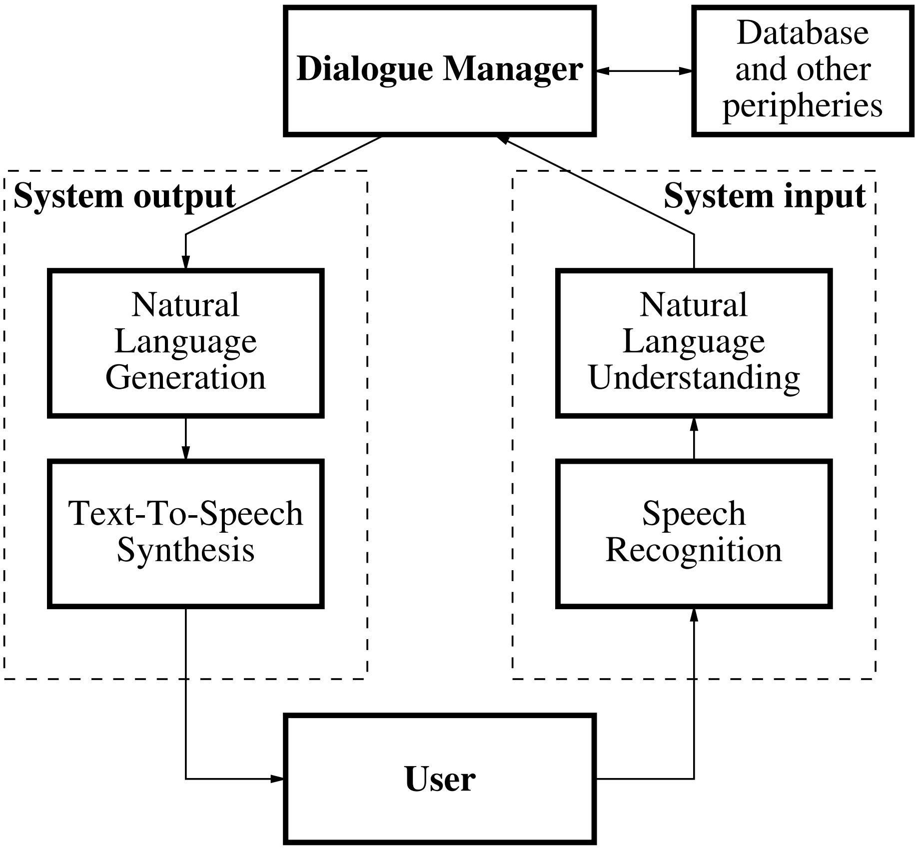


Figure 1.1 *Block diagram of a typical spoken dialogue system.*

1.2 Spoken dialogue systems

While the work in this thesis is applicable to dialogue systems using any method of communication, the specific focus is on spoken dialogue systems, where the presence of speech recognition and understanding errors poses challenges that may not be as pertinent in other types of dialogue. This section provides a brief introduction to typical systems of this type and some of the challenges involved in creating them.

1.2.1 Typical architecture

The basic components found in a typical SDS are illustrated in Figure 1.1. The diagram depicts the flow of information from the human user, to the system and back again. The essential components are subsystems for input (conveying information from the user to the system), dialogue management (deciding how to react) and output (conveying information from the system back to the user).

1.2.1.1 System input

Spoken dialogue systems typically make use of Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) systems to convert the input speech signal first to word strings and then to convenient abstract representations of information. The imperfect performance of such systems, however, makes the construction of a robust dialogue system incredibly difficult. The problem is compounded by the fact that the system must be able to operate under conditions of normal conversation, where background noise, poor articulation and incomplete or ungrammatical sentences are common. This makes recognising and understanding what has been said a problematic issue: even after a recogniser has been customised for a given dialogue application, concept error rates[1] are often in the region of 20%.

1.2.1.2 Dialogue management

The central component of a dialogue system is the dialogue manager. This is the component that coordinates the flow of the dialogue by deciding on the actions to be taken by the system. It also manages interaction with the external world through utilities such as databases or other objects the computer has access to. Deciding which action to take next is not easy, as there are typically several "sensible" options to choose from. For instance, suppose the user has input the name of a town, and the system has recognised it as such. The system has also inferred that this is the destination about which the user wishes to obtain train timetable information. Should the system ask for confirmation that it has heard the name correctly, with the risk of annoying the user, or should it go ahead and dispense information, even though the name might have been misrecognised?

1.2.1.3 System output

The final requirement of a dialogue system is some way of communicating information back to the user. This is a critical problem, since badly presented output may result in the system being perceived as stupid, confusing, cumbersome or even rude. For a fully speech-driven system, the output is again done by means of speech, using subsystems for natural language generation and speech production. Natural language generation can also be performed by simply using fixed phrases chosen for the situations that may occur, while speech production may be done using either text-to-speech synthesis or pre-recorded speech segments.

1.2.2 Applications

The literature contains numerous examples of applications for which dialogue systems have been designed. These include:

[1] *Concept error rates* are measured by calculating the frequency with which the concept being articulated is misunderstood, irrespective of the number of individual words that are recognised correctly.

- General information dispensing (train timetables, flight information, location of restaurants or hotels, directory enquiries, weather forecasts, internet or email access, etc.) [1, 2, 3]
- Negotiative tasks such as route planning, navigation or appointment scheduling [4, 5]
- Transactions such as banking, shopping, conference registration or call steering [6, 7]
- Command and control of systems ranging from light switches to robots, helicopters or space ships [8, 9]
- Teaching/tutoring (i.e. detecting gaps in knowledge or understanding of which the user may not be aware) [10]
- Informal social conversation [11]

Clearly the possibilities are endless. The main focus in most of these examples is on information gathering (collecting information from the user in order to be able to carry out a task) and information dispensing (acting as an interface to a database), but the more complex tasks also involve aspects such as inference of the beliefs or desires of the user.

1.2.3 Challenges in dialogue system design

A number of problems in the field of spoken dialogue systems are regarded as major obstacles or open questions. These include:

Portability: Although systems have been implemented for many specific applications, it remains a huge task to port a system to a new application domain. The design of the dialogue manager is a particular problem in this respect, since every new task presents new, unforeseen situations where the dialogue manager has to decide what to do.

Optimisation: Ideally, systems should be designed in a principled way so that they perform as well as possible. In the case of SDS, commonly suggested performance criteria are dialogue success rate and average dialogue length. However, it is by no means obvious how to optimise a system using these criteria.

Evaluation: There is no standard method for evaluating a dialogue system or even comparing the performance of different systems. It is impossible, for instance, to feed a standard corpus of utterances to a dialogue system: a dialogue is the result of an interaction between the user and the system, not just of the system reacting to isolated user utterances. Thus a full system evaluation can only be done by letting the system interact with real users, which is of course expensive and time consuming.

1.3 Automatic design

Given the problems mentioned in Section 1.2.3, it would clearly be desirable to automate the design process as much as possible. Since this would facilitate the design process, it would alleviate the portability problem. In addition, an automatic process could offer the possibility of optimising various parameters which human designers would normally only be guessing at. Thus design automation may offer solutions to the optimisation problem.

Of course it is overly optimistic to expect the entire design process of an SDS to be automated at once. This thesis focuses on ways in which aspects of the design of the dialogue manager may be automated, without considering the design of the other system components.[2] The basic approach relies on a representation of the dialogue as a finite state process, where the dialogue manager is a finite state automaton interacting with an environment for which a model needs to be constructed.

There are two main contributions in this thesis: first, a novel system is presented that allows simulation and evaluation of dialogue. Next, this system is used in a novel approach to the problem of automatic dialogue strategy learning.

1.3.1 Dialogue simulation and evaluation

The "environment" of a dialogue manager includes the user it is communicating with and the communication channels (including the speech recognition and understanding subsystem). In the research reported here, these components were modelled probabilistically in order to develop a system for producing quantitative dialogue simulations. This addresses the evaluation problem, since it allows statistics such as average dialogue success rate and average dialogue length to be measured without conducting expensive user trials. These measures can then be combined to obtain an indication of overall system performance, using a performance measure defined with the specific application in mind.

The relevance of the simulation system is twofold:

1. It can be used as a tool for quick and easy evaluation of prototype dialogue systems, allowing designers to detect problematic areas of the dialogue early and cut down on the number of user trials required during development.

2. It forms a component of the automatic design system, allowing the application of learning algorithms that have not been applied to this problem before.

[2]Ideally, all system components should be designed simultaneously, taking each other into account. Integrating the methods investigated here with the full system design process is thus an important topic for future research.

1.3.2 Strategy learning

A dialogue manager can be described by specifying its *dialogue strategy*, i.e. which action it chooses in each possible set of circumstances.[3] The aim of dialogue manager design is to choose a dialogue strategy that will yield good system performance. In automatic design, machine learning techniques are used to find such a strategy.

The core idea in automatic strategy learning is to use a finite state representation of the dialogue. This implies the definition of a state space, with a finite number of variables (restricted by computational requirements) describing the exact dialogue state. Transitions from one state to another are determined by the actions taken by the dialogue manager (e.g. asking a specific question), and the non-deterministic response of the environment (e.g. answering the question and correctly recognising the answer).

Once such a state space has been defined, exact descriptions of dialogue strategies can be produced. The problem then becomes one of searching through the space of possible strategies to find one that yields good (ideally optimal) performance. This thesis presents research into the application of strategy learning algorithms that make use of the dialogue simulation system mentioned above (Section 1.3.1).

1.4 Thesis outline

The rest of the thesis is organised as follows:

Chapter 2 provides an overview of the relevant aspects of the field of interactive dialogue systems, placing emphasis on issues that motivate the approaches taken here in dialogue simulation and automatic dialogue manager design.

Chapter 3 presents a framework for description of human-machine dialogue, introducing some important concepts to be used in the work that follows. This includes a preview of the methods used for modelling user utterances in Chapter 4 and a brief survey of reinforcement learning, which is used in Chapter 5.

Chapters 4 and 5 give detailed descriptions of the research performed relating to dialogue simulation and strategy learning respectively.

Finally, Chapter 6 concludes with a discussion and some suggestions for future research.

[3] Dialogue managers may also be non-deterministic, although unpredictability is not regarded as a desirable trait in most applications. The research presented here is restricted to deterministic strategies.

2

Survey of dialogue systems

2.1 Overview

This chapter provides a short introduction to the general field of interactive dialogue systems, with emphasis on issues related to the design and evaluation of the dialogue manager component. The reader is referred to [12, 13, 14] for reviews emphasising other aspects of the field.

Section 2.2 introduces a number of factors that influence the difficulty of the task being attempted and distinguish different types of system. Section 2.3 contrasts some basic approaches to implementing a dialogue system, based on different underlying models of dialogue. Section 2.4 then discusses aspects of system architecture, describing existing approaches and current research to issues in speech input, dialogue management and speech output. Section 2.5 takes a more detailed look at the problem of evaluating the performance of a dialogue system, discussing standard approaches to data collection and performance measures before reviewing the new field of evaluation by simulation (Section 2.5.3.2), which is also the topic of Chapter 4. Finally, Section 2.6 mentions a few fixed solutions to dialogue management problems before reviewing the emerging field of automatic strategy learning, which attempts to solve these problems in a principled way. This is the central topic of this thesis, and particularly Chapter 5.

2.2 Factors defining system types

This section describes some parameters that define different specifications for dialogue systems. The parameters chosen for a given system will depend on the application and the desired level of sophistication.

2.2.1 Dialogue types

In the analysis of human-human dialogues, distinctions are commonly made based on whether the beliefs and goals of the participants are shared. In particular, the following dialogue types can be identified [15]:

Cooperative dialogues: Participants have a common goal, and also share their beliefs about the topic of conversation. This is the simplest type of dialogue, and encompasses what is known as "traditional" task-oriented dialogues. Examples include information access, simple transactions such as product purchase or banking, and command-and-control applications.

Collaborative dialogues: Participants have a common goal, but do not necessarily share beliefs. Thus it may be necessary for one participant to convince the other of the "correct" belief. This will typically complicate both the structural and linguistic aspects of the dialogue. Examples include instructional dialogues between a student and a tutor, such as in [16].

Conflictive dialogues: Participants have conflicting goals, and need not share beliefs. The presence of conflicting goals causes this to be a particularly complicated type of dialogue, containing possibilities such as deception and topic avoidance. Examples include bartering in a purchasing transaction.

There are also less easily classifiable dialogues, such as social dialogues, where the participants do not have easily definable goals.

While it is possible to attempt handling the more complicated types of dialogue in text based systems, current technology has not yet advanced to the point where speech based systems are commonly attempted for applications more complicated than the basic cooperative dialogue. However, cooperative dialogues are very relevant for the types of task that computer systems are typically envisaged as being useful for. Most applications involve information retrieval and/or giving the system simple instructions, which can be done in the context of a cooperative dialogue. For these reasons, cooperative dialogues form the focus of the research discussed here.

2.2.2 Modality

Although the work discussed here deals specifically with systems that use speech for both input and output, the concept of a dialogue system can be generalised to include systems that use one or more other means of input and/or output. A medium for input or output (such as speech, gestures, touch screens or visual displays) is referred to as a *modality*, and a system that uses more than one modality is referred to as a multi-modal system. Issues such as which modalities to use for given applications and how to combine different modalities receive much attention in the literature (e.g. [17]).

For a multi-modal system, the system architecture of Figure 1.1 needs to be adapted only by replacing the speech-specific components with components that handle input or output for the modalities used in the system. Thus, the basic structure is still that of an input subsystem, a dialogue management subsystem and an output subsystem, with the dialogue manager not being dependent on the type of modalities used to communicate with the user. While the work discussed here has been done with speech-based dialogues in mind, it focuses specifically

System:	Would you like to hear your account balance?
User:	No.
System:	Would you like to do a transfer transaction?
User:	Yes.

(a)

System:	Please say balance, transfer or stock quote.
User:	Transfer.

(b)

System:	How may I help you?
User:	Transfer transaction.

(c)

System:	How may I help you?
User:	Can I, um, I'd like to transfer some money please.

(d)

Figure 2.1 *Illustrative dialogue segments for different levels of input sophistication: (a) Yes/no. (b) Isolated words. (c) Fixed phrases. (d) Natural language.*

on the dialogue manager and is therefore also applicable to systems using other modalities.

2.2.3 Sophistication of input

SDS are often classified according to the sophistication of the input the system can handle. These can range from simple yes/no answers, through isolated words selected from a short menu, connected word phrases taken from a limited list, to natural, continuous speech taken from the user's own vocabulary. Figure 2.1 illustrates dialogue segments from each of these types of dialogue. While the simpler of these types of inputs have less strenuous requirements for automatic speech recognition and are thus less prone to recognition errors, they increase the number of dialogue turns required while decreasing the naturalness of the interaction. It is therefore desirable, for the purpose of constructing an efficient, easily usable system, to support as sophisticated an input style as possible.

2.2.4 Initiative

Another important aspect defining the capability of a system is whether it allows the user to take control of the dialogue. In primitive systems, the user is expected to answer queries by specifying the information that has been prompted for, and no further information. Thus the flow of the dialogue is controlled solely by the computer, which keeps the initiative throughout the conversation. This approach can lead to dialogues which are both tedious and unnatural. An alternative, more sophisticated approach, is for a system to allow the user to take the initiative at certain points in the conversation by supplying information that has not been prompted for yet, but which either supplements or replaces the information requested in the prompt. Such a strategy is referred to as *mixed initiative*, and is

System:	How may I help you?
User:	I'd like to do a transfer.
System:	How much do you want to transfer?
User:	200 pounds.
System:	From which account do you want to transfer?
User:	From my savings account.
System:	To which account do you want to transfer?
User:	To my checking account.

(a)

System:	How may I help you?
User:	I'd like to transfer some money to my checking account.
System:	How much do you want to transfer?
User:	200 pounds from savings.

(b)

Figure 2.2 *Illustrative dialogue segments for (a) system initiative and (b) mixed initiative dialogue.*

considered desirable as it enhances the functionality available to the user and allows dialogues to be shortened significantly. Figure 2.2 illustrates the difference between system and mixed initiative dialogue strategies.

A third approach can be conceived, where the user is not normally given explicit guidance, being expected to assume the initiative all or most of the time. However, such an approach would not only present technical problems, but also require specialised knowledge on the part of the user, so that it is hard to imagine the system meeting basic requirements of ease of usability. For this reason, attention is focused on mixed rather than user initiative systems.

2.3 Models of dialogue

The theory of human-human dialogue has a long history in fields such as linguistics and artificial intelligence. The following distinct models have been applied to the construction of dialogue systems [18]:

Dialogue grammars: This is a traditionally common theory of dialogue, based on the concept of *dialogue acts*. Essentially, every utterance is classified as a particular type of dialogue act. Different taxonomies of dialogue acts exist, but examples may include information requests, answers, confirmation requests, etc. The dialogue grammar then defines rules for how dialogue acts are allowed to follow each other (for instance a question is required to be followed by an answer). Although the literature contains some examples of dialogue systems based on dialogue acts and dialogue grammars [19], they are not very common.

Plan-based dialogue: In this model, each dialogue participant is modelled as an agent having explicit goals, basing their acts on their respective goals and adapting their goals based on the inferred goals of the dialogue partner.

Joint activity model: Finally, there is the newer approach of regarding dialogue as a joint activity between the dialogue partners, with both partners attempting to optimise the flow of the dialogue as a whole.

The plan-based model referred to above has been used in systems for human-computer dialogue [20, 21, 22]. However, there are a number of arguments against grammar- and plan-based approaches [23], such as the impracticality of inferring user intentions in the course of the dialogue. Because of this, and the fact that the robustness of these approaches has yet to be demonstrated, structured approaches to system design are more common and will form the focus of this thesis.

2.3.1 Structured dialogue systems

In this approach, a rigid structure is defined for the dialogues of interest, by foreseeing different possible situations and specifying how the system should act in each of them. The approach can be viewed as making use of the joint activity model: the system designer attempts to optimise the dialogue as a whole in terms of parameters such as expected dialogue length and failure rate. The disadvantages of this class of approach flow from the fact that each possible type of situation the system may face has to be handled explicitly. This can lead to potentially inflexible systems, since many situations will have to be classed together and treated in the same way. However, structurally based systems tend to be robust, because they emphasise dialogue strategies that optimise the efficiency of communication under adverse conditions. An example would be confirmation strategies that guard against recognition/understanding errors at crucial points in the dialogue, possibly with the help of confidence measures output by the speech recogniser.

The behaviour of a dialogue manager using the structural approach is specified by a *dialogue strategy*, which specifies the action to be taken under any given set of circumstances. Most strategies lend themselves to description by means of a finite state machine (FSM), where the states of the FSM correspond to the dialogue circumstances. These are characterized by the history of the dialogue (from the system's point of view[1]) and possible external information states such as database feedback, and will be referred to as the dialogue *state*. The dialogue manager need not necessarily enumerate all states, as long as it links each state unambiguously to a corresponding system action.

[1] When recognition errors take place, this will differ from the user's point of view.

2.4 Approaches to system architecture

Irrespective of which model of dialogue a system is based on, the basic components are similar. This section discusses aspects of some system components in more detail, with reference to speech input, dialogue management and speech output. Although the discussion on dialogue management focuses mainly on systems using structural approaches, similar strategies can be implemented using plan-based approaches.

2.4.1 Processing speech input

2.4.1.1 Recognition and understanding

One of the problems in spoken dialogue systems is that of speech recognition errors, which make it hard to develop a robust system. However, it is possible to use knowledge of the dialogue domain and context to improve recognition performance.

In particular, if the system maintains an explicit representation of the dialogue state, the language model used in the speech recogniser can be made state-dependent. Various examples of this approach can be found in the literature [24, 25, 26, 27, 2].

Language understanding is typically performed by means of handcrafted grammars which are used to parse the recognition output. These grammars are designed so as to cover the range of utterances users are likely to come up with in the dialogue, thus encoding the required domain-specific linguistic knowledge. The grammars may also be state-specific, using dialogue context to improve understanding performance. The use to which dialogue context is put ranges from predicting which user utterances are more likely to more complex aspects of interpretation, such as anaphora resolution or deciding whether silence indicates the end of an utterance or merely a pause [28].

Since construction of the language understanding grammars is a time-consuming and expensive part of system development, it makes sense to use the information encoded in them to improve the speech recogniser. One such approach is [29], which uses context free grammars from the language understanding system to train n-gram language models for the speech recogniser.

A more common approach is to combine the speech recognition and language understanding processes into a single system, using a tagged finite state word network as language model for simultaneous recognition and understanding (see [30] for an early example). In this way the users' choice of expressions is restricted in order to trade flexibility of user input for robustness against recognition and understanding errors.

Research is also being done in methods that reduce the amount of work required to construct the grammars in the above-mentioned approach. For instance, [31] combines a limited number of handcrafted grammars for in-domain concepts with more complete language models trained automatically on general corpora.

Taking the idea of restricting the users' choice of expression to the extreme, [32] argues for a standardised interface that forces the user to make use of universal keywords and interaction guidelines. This allows significant improvement in recognition and understanding performance, while feedback from users is reported to be positive. However, the commonly held view is that users should be free to use natural language if the dialogue systems are to become commercially feasible.

There are also different ways in which the recognised speech can be passed on to the rest of the system. A simplistic approach is to send only a single recognition output for semantic interpretation. However, performance can be improved by the commonly used method (e.g. [33]) of including a recognition confidence measure, or even by making use of more complicated recognition outputs such as N-best lists or lattices.

2.4.1.2 Interpretation of semantics

Once a semantic representation of the user input is available, it is by no means decided how the system should interpret it in terms of setting its internal state variables.

One issue is the resolution of user utterances that are ambiguous or contradict information obtained earlier in the dialogue. The popular approach of using the most recent user input as the definitive one is being questioned [34], while various ways of using dialogue context and domain-specific information for ambiguity resolution are being investigated [34, 35].

The use of belief networks to resolve utterance meaning is an interesting approach that has recently received attention [36, 37]. In more complex dialogue, it can be useful to apply the belief network approach to infer both dialogue acts and domain dependent task goals, which can then be used during utterance resolution [38].

Work is also being done on automatic learning of out of vocabulary words and phrases, using transformation-based parsing [39] and latent semantic analysis [40, 41].

2.4.2 Dialogue management

2.4.2.1 Form-filling dialogues

One of the simplest types of structured dialogue is the so-called *form-filling* dialogue (also called *frame-based* or *slot-filling* dialogue). Systems based on forms have been implemented by various researchers [42, 43, 44], with one of the first such systems dating from as early as 1977 [45]. The approach is useful in applications where the user needs to communicate a set of specific pieces of information that can be represented in an attribute-value structure. In this context such a structure is referred to as a *form*, while the individual attribute-value pairs are called *slots*. Despite being somewhat inflexible, these systems are popular

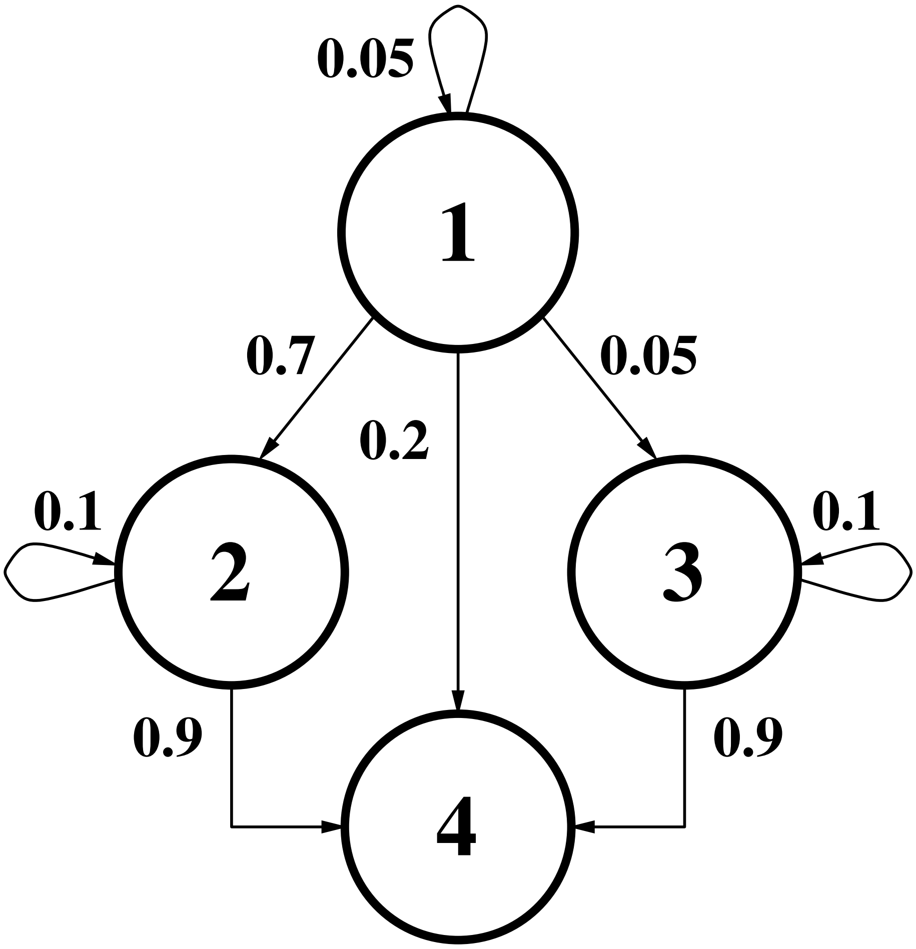


Figure 2.3 *Example structure of a form-filling dialogue.*

because of their simplicity and relative ease of implementation, with a number of toolkits being available for constructing them.

As an example, consider a dialogue where the user has to input the day and month of a date. This is done by filling in two slots corresponding to these attributes in a form. Typically, the system would start with by prompting the user for one of these variables. In a mixed initiative system, the user would then have the option of supplying the variables in any order, or both at once. The states in this example might be defined as follows:

State 1: Neither month nor day have been specified

State 2: Month has been specified but not day

State 3: Day has been specified but not month

State 4: Both month and day have been specified

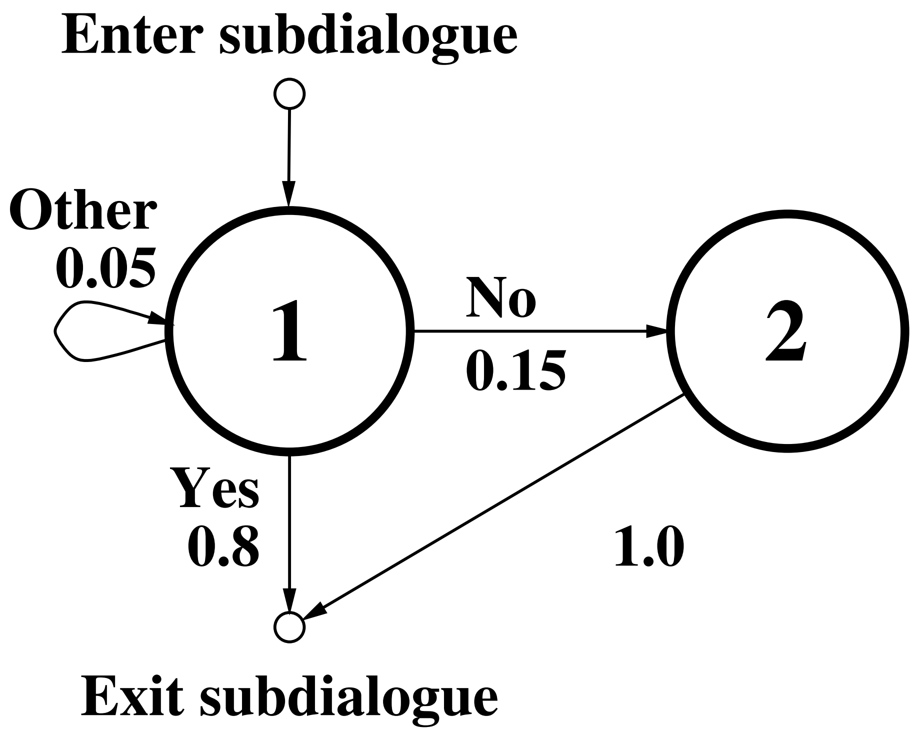


Figure 2.4 *Example structure of a confirmation sub-dialogue.*

Transitions between the states happen when the system prompts for the value of an empty slot, a user response is input and the new state is determined after interpreting the user response. Figure 2.3 illustrates a possible structure for this dialogue, with the numbered circles representing the states defined above and the numbers shown on the arcs indicating hypothetical transition probabilities.

In the presence of recognition errors, the dialogue strategy in this example would be rather poor, since no attempt is made to verify the correctness of the slot values. One way to adapt it would be to traverse a subnetwork each time the user has specified some value, before carrying on to the next state. Such a subnetwork might consist of a confirmation sub-dialogue such as the one shown in Figure 2.4, implementing the following strategy:

State 1: Output contents of most recently filled slot. Output yes/no query for confirmation of correctness.

State 2: Clear contents of most recently filled slot.

This type of strategy, using explicit confirmation, is popular when the confidence in the correctness of the slot value is low. It is also possible to use an implicit confirmation strategy, which decreases the expected length of the dialogue, but increases the probability of a confirmation failure. The issue of confirmation strategies is one that has been studied in depth by various researchers (e.g. [46, 47, 48]), and illustrates how different strategies may be optimal under different circumstances or different optimality criteria.

Real applications are of course more complex than the artificial example discussed above. In general, filling some slots may be optional, and the availability of slots to be filled may depend on previous information supplied by the user. Also, the state space representation is often expanded to include, for example, information on the quality and origin of the user input rather than just the semantic contents.

2.4.2.2 Adaptation in dialogue management

Since structured dialogues are generally regarded as inflexible, much attention is given to ways of designing systems that can adapt to circumstances in various ways.

A proposal to increase system flexibility is that of "limited enquiry negotiation dialogues" [49]. This extends the idea of form-filling dialogues by allowing the dialogue participants to negotiate over which of the available slots to fill for a database enquiry.

A number of studies have been done on adapting the dialogue strategy to suit the user's level of expertise (for more expert users a larger amount of mixed initiative and less strict confirmation can be used). These include systems that maintain a model of user expectations and abilities [50] and systems that monitor the dialogue performance [51, 52]. Improvements of the adaptive strategy over a non-adaptive approach were verified empirically in the case of [51]. A similar idea was proposed in [53], where the word confidence threshold is adapted based on confidence levels earlier in the dialogue.

Systems can also be designed to react to events in the external world. The idea of dynamic system state is proposed in [9], where a helicopter command and control system reports on external events as they happen. Similarly, the system in [39] reacts to changes in share prices during the course of the conversation.

Another idea is user-specific adaptation of the dialogue system to cater for the preferences of individual users. For instance, [54] proposes a user-specific application development process. The application has a simple predefined structure, with specific items being configurable on the basis of profiles specified by individual users.

2.4.3 Producing speech output

A common approach to speech output production is simply to prerecord prompts for different dialogue states, using templates to insert detailed information where necessary. This is by no means trivial, since the choice of prompt can have a huge effect on user behaviour.

More recently, progress has been made in automatic approaches to sentence generation, aiming to produce more natural sounding output. This is likely to be useful, since finer aspects of dialogue state are hard to take into account using a template-based approach.

A recent system with an automatic sentence generator that can be trained using human feedback is presented in [55]. The training mechanism addresses the portability problem by allowing for domain specific customisation. Other research focuses on increasing adaptability of system output, such as [56] which uses informational novelty to vary word order and [57] which uses both informational novelty and confidence levels to adapt the degree of explicitness of the output.

2.5 System development and evaluation

The standard process of developing a dialogue system consists of several *development cycles*, each of which consists of a design stage followed by an evaluation stage. The evaluations are aimed at pointing out problems in the prototype system, which are then addressed during the next design stage. This section discusses issues in system evaluation, both as a part of the development process and as a means of measuring the performance of the final product.

2.5.1 Data collection

The first step towards evaluating a system is to collect a corpus of data on which the evaluation can be performed. In contrast to a problem such as speech recognition, it is not possible to collect a general data corpus for comparative evaluation of different systems: the data produced in a dialogue depends on the strategy and performance of the system being tested, and the effect this has during interaction with the user. Thus a new data set is required for every new application being developed and tested.

Once a data set is available, it can be used for evaluation purposes by measuring various performance-related parameters, as well as for design purposes by observing which syntaxes are used by users, where the problem areas in the dialogue are, etc.

A number of methods are used for acquiring dialogue data:

System in the loop: Tests are run by letting the prototype system under development interact with human users. Results on each such test are used to optimise the system, yielding a new prototype system so that the entire process can be repeated. The loop continues until the system has been improved to a point where results of a test are satisfactory.

The obvious drawback of this method is that data collection experiments have to be performed numerous times during the development process, making it expensive and time consuming. Also, users will react differently to an imperfect system than they would to the eventual, finished product (see [58] for an example of a study of how users adapt their utterances to a specific system). This means that the collected data sets may be skewed and contain user behaviour that is inappropriate for the final product.

Wizard of Oz simulation: The "Wizard of Oz" method [59] refers to an experiment where a dialogue system is impersonated by a human operator, who interacts with prospective users. In this way, a data corpus is collected which can then be used to extract parameters for use in system development. The advantage of this method is that, if the impersonation of the eventual system is realistic, the measured user responses are those that would occur with a finished system, rather than an imperfect prototype system. Thus, repeated data collection should not be necessary. In practice, this is often used as a first step prior to moving over to "system in the loop" development for the later stages. The method suffers from the disadvantages that data collection is still expensive and time consuming, and that impersonation of a working system often tends to be unrealistic and unreliable.

More detailed descriptions of system development using the Wizard of Oz technique can be found in [60, 61].

Computer simulated data: An attempt to sidestep the requirement for expensive data collection trials is to simulate dialogues automatically. While simulated data cannot be expected to be highly accurate, it can nevertheless prove useful for the development process: intermediate performance evaluations can be done using the artificially simulated data, so that the number of tests and/or data collection experiments to be conducted are reduced significantly. Evaluation by simulation is discussed in Section 2.5.3.2.

2.5.2 Performance measures

Apart from intermediate evaluations that may be performed during the development process of a dialogue system, performance evaluation of the final product is of the utmost importance to decide whether it is fit for commercial use. However, there is no clear, objective method to perform such an evaluation: the eventual value of a system depends on the opinion an end user will form and is therefore highly subjective and dependent on the group of people who will use it. In spite of this, developers have to attempt an approximate evaluation.

Numerous objective and subjective quantities have been used for system evaluation. Subjective measures are typically obtained by asking users to award satisfaction ratings to system aspects, such as ease of use, friendliness, efficiency (as experienced by users), system understandability or appropriateness of interaction pace. Objective measures are calculated from data collected during evaluation, and include task success rate, task duration in seconds, average number of dialogue turns, word error rate and number of incorrect system actions.

It is not at all obvious which of these measures are the most appropriate or how they can be combined in order to obtain a single metric for system performance. Proposed approaches include the EAGLES [62] and PARADISE [63, 64] paradigms.

The EAGLES approach is to measure a number of independent metrics which can then be used for diagnostic purposes. The evaluation is divided into “black box” evaluation, where external performance metrics such as task success rate and turn duration are measured, and “glass box” evaluation, where internal system performance metrics such as word accuracy and concept accuracy are measured. The approach does not attempt to obtain a single performance measure by combining the metrics.

The PARADISE approach goes a step further by constructing an evaluation function that is a linear combination of objective measurements such that it correlates maximally with user satisfaction, as measured by polling users at the end of the dialogues in a system trial. This is very useful since it provides a single metric that can be optimised, for instance by the automatic learning algorithms discussed in Section 2.6.2. Unfortunately, it is not clear how much can be concluded from the results of applying this method, as different experiments do not seem to reach similar conclusions.

For example, [65] found that user satisfaction depends on the order in which different systems were presented to the user rather than on the system parameters. In [66] little correlation was found between recognition performance and user satisfaction ratings, [67] found that elapsed time is not an important predictor of user satisfaction (although its importance increases as task success rate improves), and [4] found that task completion, task duration and sentence recognition accuracy are significant predictors of user satisfaction. In both [6] and [68] there was a negative correlation between user satisfaction and dialogue length, explained by users hanging up when they are dissatisfied. This means that the causal link between dialogue length and user satisfaction is in the wrong direction: the correlation cannot be utilised to optimise user satisfaction.

For these reasons, evaluation functions are often constructed in an ad hoc way by system designers, from objective measures such as dialogue success rate and number of dialogue turns. For instance, [69] defines metrics called “percentage of confused users” and “percentage of satisfied users”, as respectively the percentage of dialogues where the system asks confirmation questions about incorrect information and the percentage of dialogues where the transaction is completed correctly while maintaining a fixed minimum slot transmission efficiency. Similarly, [68] defines a metric called “error correction metric” which measures the number of times the user has to repeat a given concept in the course of completing a task.

2.5.3 Evaluation without system trials

Because of the time and expense involved in conducting user trials, any method for evaluating a system without such a trial is highly desirable. Even a rough approximation is likely to be useful in speeding up the development process.

2.5.3.1 Mathematical analysis of dialogue structure

Within the structural approach to dialogue design, work has been done to evaluate different strategies for certain common substructures in dialogue by means of mathematical analysis [70, 71, 72, 73]. In particular, different strategies for confirmation of information input by the user have been analysed in order to see how the optimal strategy changes for different recognition accuracies and optimality criteria. In some cases, it is possible to obtain closed form expressions for metrics such as the number of dialogue turns and dialogue success rate. Unfortunately, the analysis can only be applied to comparison of a few proposed strategies, which motivates a simulation method that can be used to simulate any implemented system.

2.5.3.2 Simulation systems

Computer simulations of dialogue are not new: simulation has long been used as a tool to investigate theories of dialogue. Early work in this field was done by Power [74], who implemented a framework in which two computer agents engage in dialogue, using the concept of *dialogue games*. Other research in this direction has been done by Houghton [75], whose *interaction frames* are a form of dialogue game theory, and Hovy [76], who investigated the effect of pragmatic goals and the different type of goals agents might have. Carletta [77] implemented a plan-based system that made use of detailed belief models and agent goals.

While these simulation systems had parameters that could be changed to produce different agent behaviours, the parameters needed to be set by hand rather than trained from data. Also, the simulation was purely deterministic, so that the dialogue output depended solely on the parameters and states with which the agents were initialised. This is satisfactory for the purpose of qualitative evaluation of the fidelity of the simulation, which is what is required when judging a particular theory of dialogue. Unfortunately, it is not appropriate for the purpose of evaluating the performance of a dialogue system, where a full spectrum of the dialogues that can result from a given set of parameters and initial states has to be generated.

A number of researchers have proposed using probabilistic computer simulations to evaluate dialogue systems:

In an early study, Hone and Baber [78] used a simulation to compare transaction times for two versions of a banking system. The user was constrained to answering specific queries with no more than one concept per user utterance (no mixed initiative) and could thus be modelled by assuming that s/he always answers the questions directly and correctly. Errors were modelled using a fixed recognition accuracy, with the assumption that all errors are substitution errors and all substitutions are equally likely.

Araki and Doshita [79] proposed an evaluation environment using system-to-system dialogue (as opposed to system-to-human dialogue) with addition of linguistic noise. The linguistic noise is realized by generating either deletion, insertion or substitution errors at the word level, according to a fixed word error rate. Unfortunately, the environment requires the availability of implementations of both dialogue partners. This is of course a problem, as the goal of a dialogue system is to interact with humans rather than other systems.

Eckert, Levin and Pieraccini [80, 81] implemented a data-driven model of a human dialogue partner in another automatic evaluation system. This work was conducted in the ATIS flight information domain, using a relatively simple dialogue structure for which a simple bigram user model yielded satisfactory results. However, no error modelling was incorporated into the system. This model will be discussed further in Section 4.2.2.

The abovementioned model by Eckert, Levin and Pieraccini served as a starting point for the work presented in Chapter 4, in which a more detailed data driven model of user behaviour was used, along with a data driven model of recognition/understanding errors. Descriptions of this system can also be found in [82, 83, 84].

Finally, Lin and Lee [85, 69] developed a simulation method that is not data driven. Instead, parameters are chosen by hand and system performance can then be evaluated for the chosen parameters. Errors are modelled by assuming fixed rates for substitution, insertion and deletion errors, while a limited range of fixed user response patterns can be simulated. (With no data being used, the choice of user and error models is left to the system designer.)

A simple slot-filling dialogue structure is assumed, where the aim of a transaction is to fill and verify all the slots. Although a single, simple dialogue state representation is chosen, this can in principle be extended to representations of arbitrary sophistication. The system works by investigating a limited range of possible system strategies (for a hypothetical system), rather than interfacing with an existing system. Thus it is envisaged to be used at the initial stage of system design, when it would be especially useful since no data is yet available for the specific application. However, it has the drawback that the model cannot be expected to be as accurate as one based on data.

2.6 Designing strategies for dialogue management

2.6.1 Fixed solutions for dialogue management problems

Problems encountered in dialogue manager design include selecting strategies to determine the style of confirmation, use of confidence levels, and the amount of mixed initiative provided to the user at various stages in the dialogue. Unfortunately, system designers typically have no choice but to make ad hoc decisions on these questions. Many papers suggest specific solutions that have been implemented. For example, [33] and [47] present interesting ideas on the use of confidence levels to realize adaptive confirmation strategies.

However, the question of how these specific solutions compare to the many alternative possibilities needs to be addressed. Apart from the work mentioned in Section 2.5.3.1, a number of researchers have performed experiments to compare a limited number of alternative dialogue strategies. In [86] user trials found a system initiative strategy superior to a mixed initiative strategy in a telephonic email access application, while in [46] three different confirmation strategies were compared in a train timetable information system, with a direct confirmation approach being found to work better than more indirect approaches.

A different experiment is reported in [87], where two versions of a system (allowing different degrees of user initiative) were evaluated in Wizard of Oz simulations with two distinct user groups and different levels of speech recognition accuracy. It was found that different strategies are required for different circumstances: the best strategy depended on the user group and the recognition accuracy.

The latter study suggests that choosing a fixed solution for dialogue management problems is not ideal: a method is desired to design the strategy while taking into account factors like the specific application domain, user group, recognition/understanding system and speech output system that are going to be used.

2.6.2 Dialogue representations for automatic strategy learning

The plan-based methods mentioned in Section 2.3 are one way of automating aspects of system design. However, they are particularly suited for high level aspects of the strategy. In this thesis the focus is on low level questions such as how to manage the confirmation strategy or what amount of initiative to allow the user, which are not as easily addressed by such methods.

The idea of using genetic algorithms, which are a variant of reinforcement learning algorithms, to select dialogue strategies was mentioned by Walker [88] in 1993. Subsequently, Levin, Pieraccini and Eckert [89, 90, 91] proposed an approach that is well suited to finding solutions for the low level questions of interest. First, they show that any dialogue system can be represented as a sequential decision process. The result of this is that, using a finite state representation of the dialogue, the dialogue manager can be specified as an agent to be trained using machine learning techniques such as reinforcement learning. A number of variants of this approach are discussed below.

2.6.2.1 Markov decision process with a simulated user

By making additional assumptions about the state transition probabilities and cost assignment, the dialogue system can be described as a *Markov decision process (MDP)*. The theory of MDPs is well established [92, 93], and has found many applications in fields such as machine learning and adaptive control. The standard class of algorithms used to train agents in MDP environments is known as *reinforcement learning* and depends on the ability of such an agent to experiment within its environment. A brief review of Markov decision processes and reinforcement learning is provided in Section 3.4.

In order not to create a system that has to learn in its real environment, which would require having humans available to interact with it for potentially a very long time, the proposal is that the environment be simulated probabilistically. Reinforcement learning can then be used to find an optimal strategy by letting the system interact with the simulated environment.

A detailed discussion of this technique was given in [91], while another discussion with some simple illustrative experiments was presented by Young [94].

Levin, Pieraccini and Eckert present experiments performed on a database querying dialogue application, where this approach was used to learn a strategy similar to those that human designers have implemented for the same application. The problem was simplified by ignoring recognition/understanding errors, so that no confirmation sub-dialogue was present. Mixed initiative was also not used. These simplifications allowed the assumption that the user response depends only on the current system action, so that it was possible to use a very simple user user model.

The same approach was used by Goddeau and Pineau [95] to learn optimal confirmation strategies in an artificial form-filling application. The user model was augmented with a simple error model using a fixed substitution error rate.

The work presented in Chapter 5 (see also [96] for a brief account) builds on this approach, making use of the more sophisticated simulation system presented in Chapter 4 to allow learning in more complex environments.

A different idea was proposed by Lecœuche [97], who used *inductive logic programming (ILP)* to learn rules that generalize the strategies obtained by the above technique. This can be useful for summarising the strategy in a human-readable form, so that designers can study and possibly re-use it in other applications. He also proposed using ILP to learn generalised rules at regular intervals during the reinforcement learning process, so that these rules can be used to direct the policy being followed in further learning. This approach was tested on an artificial problem, and gave a large reduction in the number of iterations required to find the optimal strategy.

2.6.2.2 Markov decision process with a state transition model

The need for a user simulation can be avoided by constructing a model of dialogue state transitions, which can be estimated directly from a corpus in which these transitions have been logged.

Systems using this approach have been developed by Walker, Fromer and Narayanan [98, 99], who applied the method to an agent for telephonic email access, and Singh, Kearns, Litman and Walker [100, 101, 102, 48], who used it to develop systems for telephonic access to train schedule information and information on local activities. Aspects of strategies investigated included different approaches to confirmation, mixed initiative and information presentation.

These experiments demonstrated that it is possible to use this method to solve design problems for complex, realistic systems automatically. A drawback of the approach, however, is that the system state representation should be chosen before data collection if the collection is to be done in a way that ensures sufficient coverage of the entire state-action space. A more detailed comparison between this approach and the simulation-based approach will be given in Chapter 5.

2.6.2.3 Partially observable Markov decision process

Representing a dialogue as a Markov decision process requires the assumption that the system can always observe the full dialogue state. However, the presence of imperfect recognition means that this is not the case. While the dialogue state can be defined in such a way as to alleviate the negative effects of making this assumption (as evidenced by the success of the approaches mentioned above), it would clearly be preferable to avoid it altogether. This can be done by describing the dialogue as a *partially observable Markov decision process (POMDP)*. The state space representation can then use the actual user goal (as well as hidden system variables), instead of just the knowledge available to the system which MDP-based methods are restricted to. Unfortunately, the algorithms available for solving a POMDP are computationally expensive, making the task of finding optimal strategies for most real dialogue systems intractable. However, a number of researchers have been attempting to overcome this problem.

Roy, Pineau and Thrun [103] proposed the use of an algorithm for approximate POMDP solution, the Augmented MDP, which involves the assumption that uncertainty as to the exact state is localised. This reduces the computational complexity from exponential to linear in the number of states, although it is still much more expensive than an MDP approach. Experiments were performed on a small (13 states) mobile robot application, using a handcrafted state transition model, with the approximate POMDP solution outperforming the MDP solution in simulations. The exact POMDP solution was intractable for this problem, but outperformed the other algorithms on a smaller version of the problem.

Another proposal by the same authors [104] is to use a hierarchical approach to decompose the problem into smaller subproblems. This was tested on the same robot application, and also succeeded in making the problem tractable (although again much slower than the MDP approach) and outperforming an MDP solution.

A third POMDP approach was presented by Zhang et al. [105, 106]. This approach uses Bayesian networks for state estimation and low-level information extraction, combining information from different input modalities. Experiments were performed on a scaled down example application, which with 40 states was still too large for exact POMDP solutions to be tractable. A number of approximation algorithms were compared, with a grid-based approximation outperforming other versions, including two MDP algorithms, in simulations.

These approaches are potentially promising, and might well replace MDP approaches. In all cases the state transition models were handcrafted, but they could in principle be obtained using data-driven techniques similar to those used for MDP approaches. Unfortunately, though, even the approximate POMDP algorithms are much more computationally expensive than the corresponding algorithms for MDPs, so that it remains to be seen how well they will scale to real world applications. Also, it is not clear whether the POMDP state space representation used is optimal when these approximation methods are used to find the solutions, so that the relatively poor results for MDP algorithms are not necessarily indicative of their performance under other state space representations. Thus it remains to be seen whether approximate POMDP approaches offer any advantage over MDP approaches.

2.7 Summary

A number of important concepts in the field of interactive dialogue systems have been introduced. In particular the structured approach to dialogue management was discussed, with examples of typical decisions that have to be made during the design of dialogue managers using this approach. The problem of evaluation was discussed, with specific reference to the idea of using simulations (data-driven or otherwise) for evaluation purposes. Finally, the field of automatic strategy learning was discussed, distinctions being made between a number of different approaches to the idea. The remainder of this thesis will present research on the simulation-based MDP approach to this problem.

3

A framework for describing human-machine dialogue

3.1 Overview

3.1.1 Dialogue as an interaction between agents

In this thesis, human-machine dialogue is viewed as a process whereby two agents interact in a common environment. The agents involved are the two conversational partners: the human user and the dialogue manager. The environment is the communication channel, which in the case of a spoken dialogue system is a noisy acoustic channel.

Both agents are attempting to optimise the interaction as a whole, and each of them can model the process as an interaction between themself and a complex environment which includes the conversational partner. For the purpose of this thesis, the viewpoint adopted is that of the dialogue manager, which is the agent that needs to be designed. Seen from its perspective, the other components of the dialogue system form part of the noisy communication channel, which together with the user makes up the environment.

The interaction between the dialogue manager and its environment is characterized by the flow of information between them. It is this information flow that gives structure to a dialogue, in terms of which it can be described. Once such a description is available, the effectiveness of the interaction can be evaluated and the agent can act so as to optimise the interaction.

3.1.2 Chapter outline

This chapter starts by presenting some concepts which are used to described the overall structure of dialogue in Section 3.2. Section 3.3 then discusses the aspects of the system environment that need to be modelled, specifically the user behaviour and the speech recognition/understanding component. The aim in modelling these components is to allow quantitative dialogue simulations to be performed, which can then be used for evaluation. To this end, a novel model of both the generation of utterances and their corruption by errors is introduced. Finally, Section 3.4 provides a brief review of the reinforcement learning techniques that will be used for automatic strategy learning, as well as a discussion on how these techniques are applicable to dialogue.

3.2 Dialogue structure

Irrespective of whether or not a structural approach is used to implement the dialogue manager, all dialogues share some basic structure. The following concepts related to dialogue structure will be useful.

3.2.1 Utterances

A dialogue consists of an alternating series of user and system *utterances*. For the purposes of this thesis, the term utterance will be used to refer to the intention (concept) level description of what either dialogue participant has said during a dialogue turn. System utterances will also be referred to as *prompts*. The structure inherent in user utterances will be discussed further in Section 3.3.2

3.2.2 Transactions

At a somewhat higher level, a task oriented dialogue can be seen as a sequence of *transactions* between the user and the system. A transaction is typically completed by the system taking some action, such as providing an item of information or performing a transfer between bank accounts. If a task oriented dialogue system is defined as one that provides the user with a set of possible tasks that can be completed, then a transaction is a section of the dialogue starting at the beginning of the dialogue or the end of the previous transaction and ending when a task is either completed or abandoned.

A transaction is completed *correctly* if the system performs the action that the user intended it to perform, i.e. that corresponds to the user goal (see Section 3.2.3). If it is completed in a relatively small number of dialogue turns (utterances), it is said to have been completed *efficiently*. It is a goal of dialogue design to create a system that enables transactions to be completed both correctly and efficiently.

3.2.3 User goals

The notion of a transaction is closely related to that of a *user goal*, which corresponds to a particular transaction which the user wants to be completed. A user goal can be defined as "an informational lower bound describing the amount of information to be acquired before the action associated with this goal can be executed" [107]. It is assumed that, for task oriented dialogues, a user goal always exists.

In this thesis, user goals will be represented by means of attribute-value structures such as that in Table 3.1. The structure contains a number of goal fields,

Goal field	Value
F1:	A
F2:	C
F3:	-

Table 3.1 *Example attribute value structure representing a user goal.*

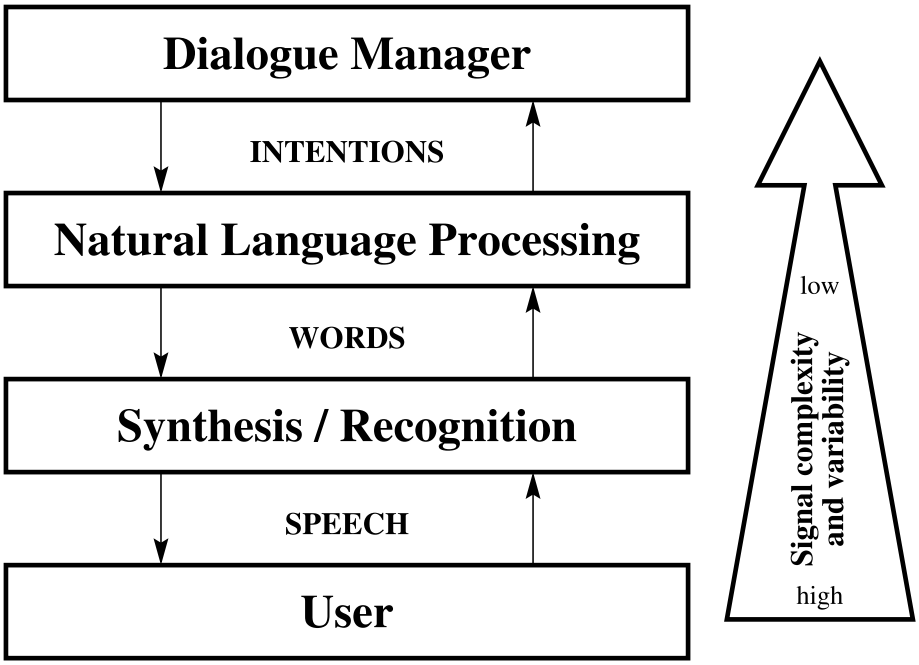


Figure 3.1 *Block diagram indicating levels of human-machine interaction.*

denoted by *F1*, *F2* and *F3* in the example. A goal field may either be uninstantiated, or bound to a one of a predefined set of allowable values. These values typically denote informational items that have a specific meaning in the context of the application. In the example, $F1$ has the value A and $F2$ has the value C, which might for instance denote a day of the week and a travel destination respectively. $F3$, on the other hand, is uninstantiated: this indicates that no information is available for this goal field.

In this thesis the values in user goals will always be atomic, but in general the framework allows the use of more complex structures.

3.2.4 Level of description

The basic components of a spoken dialogue system have already been introduced in section 1.2.1. Figure 3.1 presents another view, indicating the types of information transferred between the various subsystems. The following levels of interaction can be distinguished [80]:

Speech level: Speech is the signal that is physically transmitted from the user to the system. Thus the speech level is the most obvious interface between the dialogue participants.

Word level: One could also think of the speech recognition and text-to-speech units as part of the transmission channel between the participants. In this view, the interface is at the word level, with communication taking place in a text format. Note that the distortion in the transmission channel is increased in this perspective, as it now includes speech recognition errors.

Intention level: Taking the idea of a word level interface one step further, we arrive at the intention level. Intentions represent the actual information (concepts) that a dialogue participant wants to convey, and are analogous to the dialogue acts of classical dialogue theory. All interpretation of the physical signals is now considered part of the communication channel, allowing intentions to be thought of as being conveyed directly, and highlighting the dialogue manager as the core of the dialogue system.

If the dialogue is to be described at the speech level, it would be necessary to construct models for not only the thought processes of the user, but also the speech understanding and production processes. Similarly, a word level description would still necessitate modelling of human understanding and language generation. The latter is especially problematic, as it would necessitate mapping each possible user intention to a representative set of utterances. Given the wide range of possibilities for phrasing even a simple intention, this would be a huge obstacle.

Fortunately, it is possible to model the dialogue at the intention level, bypassing the lower levels completely. For the purposes of this thesis, an *intention* will be defined as *the minimum unit of information that can be conveyed independently within a given application*. This means that the meaning of an intention is not altered by context, so that an utterance can be viewed as a set of independent intentions, with their order of occurrence being irrelevant.

More precisely, an intention will be taken to refer to a either a specific instantiation of a goal field value, or a keyword of which the exact semantic meaning is defined in the context of the particular application. Thus, a user with the goal in Table 3.1 might produce an utterance which is intended to mean that $F1$ is to be given the value A. This constitutes a single intention which can be denoted as $F1 : A$. If $F1$ is the only field to which the value A can be assigned, the intention can be denoted simply as A. However, in applications where the value A could also be assigned to $F2$, specifying only the value would be ambiguous and the name of the goal field would also be required to make it a valid intention.

In this thesis, intentions will be indicated either by labels such as in the preceding paragraph, or, in specific examples, by informal natural language strings that describe their meaning.

The existence of unambiguous intentions in the mind of the user is hypothetical, but for the purpose of simulating dialogue it is a useful simplifying assumption. It is assumed not only that such intentions exist, but also that they can be mapped to the set of intentions used by the dialogue system. Thus two intention level representations can be constructed for a user utterance: one corresponding to the intentions in the mind of the user and one corresponding to the intentions understood by the system.

The following points argue for working solely on the intention level:

- Constructing a reasonable model of user behaviour at the word level is probably infeasible, because of the vast domain space of possible user outputs for even a simple utterance. Data sparsity would make the task of parameter estimation impractical if not impossible.
- Modelling user behaviour at the word (or lower) levels is unnecessary for the purpose of high level system design and can be replaced by simply incorporating performance statistics from the lower levels. Separating the higher and lower levels of system design simplifies both tasks.
- Results produced by working at the intention level are valid for any language, as well as non-speech modalities.
- Eliminating the linguistic component makes the simulation and automatic design procedures much more domain independent.

3.2.5 States

Another way to look at the structure of a dialogue is to analyse the sequence of *dialogue states*. It is important to distinguish between two different notions of dialogue state that are sometimes conflated:

Overall information state: This consists of all information relevant to the dialogue, including the dialogue history and any state information that is internal to the dialogue participants without being commonly known.

System state: This consists of all information known to the system and is a subset of the overall information state. Similarly, the user can be regarded as having an internal state, which includes the hidden user goal and is also a subset of the overall information state.

The research reported here is concerned with systems in which the system state is represented explicitly. In such systems the dialogue strategy can be specified as a map from states to actions, which allows the use of automatic design techniques.

Many researchers have proposed specific representations of system state (e.g. [107]), after which research focuses on designing strategies in the chosen representation, often by considering the state sequences that can be expected in typical dialogues. However, the issues of finding a good state space representation and of mapping it to a good strategy are both unsolved problems.

3.3 Modelling the environment

3.3.1 User modelling

The problem of modelling human users of dialogue systems is far from trivial. The first point to note is that humans behave very differently when speaking to a computer than when speaking to another human (see [108] for a recent study quantifying differences between human-human and human-computer interaction styles). This means that the extensive knowledge that has been built up of human-human dialogue is of little use for the present purpose: new models have to be created based on human-computer data.

Studies of human behaviour towards dialogue systems have resulted in a number of conclusions [108, 109]:

- User utterances have relatively little variability at the intention level (the amount of user initiative is often small).
- User utterances are often ungrammatical and processing of individual intentions rather than full grammatical constructs is required.
- The quality of the system output influences user behaviour.
- Users tend to get angry and act in an unpredictable way (usually out of domain).

These observations give further motivation for working at the intention level: it sidesteps the large amount of complexity at the lower levels and makes use of the relatively lower complexity at high levels. The large effect of the system prompts emphasises the necessity of building models using data collected from the system under investigation. Models that are not based on such data cannot be expected to be accurate.

Another important point is that the way users act is unpredictable, and complex enough to require modelling as non-deterministic. Thus a probabilistic user model is required to produce realistic simulations.

However, it will be argued in Section 4.2.2 that a purely probabilistic model is insufficient for dialogues with a realistic level of complexity. This is because real users do not act randomly. Rather, they base their actions on some hidden but deterministic goal and on their internal belief state, so that different actions in the same dialogue tend to be consistent with one another.

The model developed in Chapter 4 is based on the assumption that user behaviour is driven by a goal that remains fixed for the duration of the transaction. This is not always the case in real dialogues, since users often change their minds about what they want to do. However, in task oriented systems it is reasonable to assume that users should normally be attempting to accomplish fixed tasks.

The other major assumption is that the way in which users go about achieving their goals can be modelled using fixed probabilities estimated from data. Of course, user behaviour may change with time so that these probabilities are really non-stationary. However, this is alleviated by the fact that the model parameters can be re-estimated on a regular basis as more data is collected.

3.3.2 Utterance structure

In order to generate simulated user utterances, some understanding of their structure is required. Since the simulation is to be done at the intention level, the grammatical complexity of word level utterances is irrelevant: an intention level utterance is simply a set of intentions grouped together. This section presents a model for a way in which such utterances may be formed.

3.3.2.1 Choice points

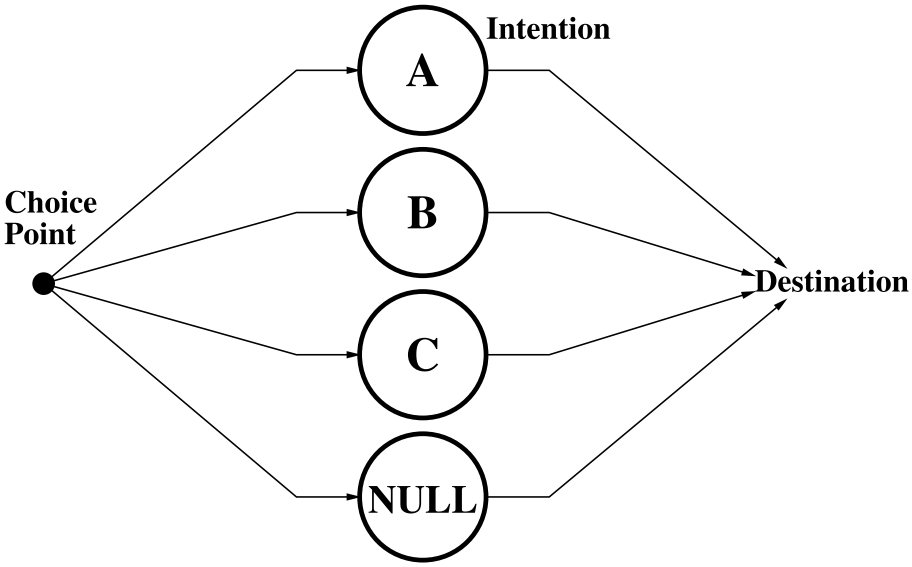
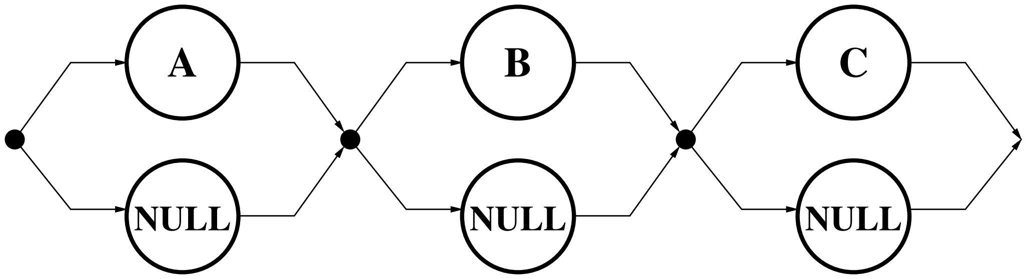


Figure 3.2 *A single choice point.*

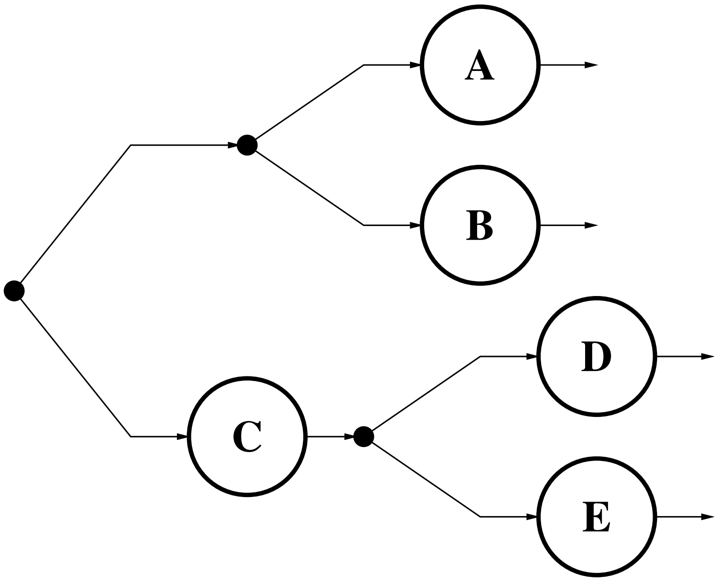
During the construction of an utterance, the user may be faced with a series of choices. For example, consider an application where the meaningful units of information that can be specified include the transaction type. The first choice might be whether or not to specify a transaction type. The decision to do so then leads to a second choice, namely which type of transaction to specify. This is followed by a choice of whether to specify any further details, and so on. The points where such choices are made will be referred to as *choice points*.

Figure 3.2 illustrates a situation where the user has four options: specifying any of three different intentions, or none of them. If an option is chosen that has an intention (represented by a lettered circle) associated with it, that intention is appended to the utterance under construction. If the option with the “null” intention is chosen, no intention is added. Since in this example all four options have the same destination, the next choice with which the user will be faced is not affected by which option is selected at this choice point.

3.3.2.2 Utterance generation grammars



(a)



(b)

Figure 3.3 *Example utterance generation grammars: (a) concatenated choice points (b) choice from different utterance types*

Choice points can be combined in a directed graph, also referred to as a finite state network or finite state grammar, to model the construction of entire utterances. Figure 3.3 illustrates some examples. In the first, the utterance is constructed by concatenation of consecutive choice points. Each choice point is entered irrespective of which choice was taken at the preceding point. This structure occurs in a situation where the user has the option to communicate any combination from a set of three intentions. The first choice is whether or not to include intention A in the utterance. After making the choice, the user has to repeat it for intention B and then C. This kind of utterance is common in form filling parts of mixed initiative dialogues, which provide the option of specifying extra details that have not been prompted for. The second example shows a slightly more complicated example, where the user has completely different types of utterances to choose from. An example from a real application will be presented in Section 4.3.4.

This model of utterance generation is very useful for systems that use finite state networks as language models for recognition. Firstly, such systems already have grammars available since they have been designed for the recogniser. Typically these grammars are tagged word level networks, that need only be converted to the intention level by removing lower level detail. Secondly, since the user utterances are parsed through these grammars during understanding, they appear to the system to have exactly the structure proposed in the model. Thus, from the point of view of the system, the model is completely accurate.

3.3.2.3 Deterministic and probabilistic choice points

If the assumption is made that the user has a goal consisting of a fixed set of intentions to communicate in the course of the dialogue, a model of utterance generation results where a distinction can be made between two types of choice point. The first type arises if the intention “A” in Figure 3.2 is one of the intentions that has to be communicated, while “B” and “C” are intentions that are in conflict with “A” and should not be communicated. For instance, “A”, “B” and “C” might be different travel destinations from which the user is selecting. In this case, knowledge of the user’s goal is all that is required to predict the choice. This type of choice point will be referred to as a *deterministic choice point*.

The second type of choice point arises when all of the available options are consistent with the user’s goal. For instance, “A”, “B” and “C” might be, respectively, a travel date, a departure location and a destination, of which the system is only prepared to accept one in this utterance. Now the user’s choice might be different on different occasions, depending on subtle factors that cannot be predicted. The best that can be done is to model the user’s general preferences probabilistically by observing a large number of cases: perhaps in this situation “A” is specified 50% of the time, each of “B” and “C” 20% of the time, while for the remaining 10% none of the intentions is specified. This type of choice point will be referred to as a *probabilistic choice point*.

3.3.3 The communication channel

The second part of the environment with which the dialogue manager interacts and which needs to be modelled in order to simulate dialogues is the communication channel. While it is reasonable to assume that communication from the system to the user is free of errors, this is not the case for communication from the user to the system. Errors in speech recognition and understanding need to be modelled in detail, especially if the model is to be used to optimise fine aspects of dialogue strategy. This may include treating confirmation of different intentions differently, depending on the relative probabilities of confusing them. For this reason, the error model should be more sophisticated than a single global error rate.

The problem of modelling the effect of the input system is made surmountable by grouping the different error types together and considering only the total distortion that takes place between the original intention in the mind of the user and the eventual interpretation arrived at by the system.

The choice point model of utterance generation also allows modelling of recognition errors. The idea is that, after the user selects option “A” in Figure 3.2, it may be misrecognised as, say, “B”. The probability of each possible misrecognition of this type can be estimated from data, either in the particular dialogue context, or, when data is sparse, from all occurrences where confusion of these two intentions were possible.

The null intention is used to facilitate the modelling of insertion and deletion errors in the production of the utterance: if the user’s choice was to select the option “A”, deletions are governed by the probability for it being misrecognised as the null intention. Similarly, an insertion is simply a misrecognition of the null intention as a normal intention.

A choice point does not necessarily have an intention (null or non-null) for every option. This complicates the issue of generating errors via substitution somewhat, since it has the effect that intentions can sometimes be substituted by intentions at a different choice point. In the second example of Figure 3.3, for instance, intention “A” may be substituted by either of “B” or “C”, though not by “D” or “E” as those intentions could not have occurred without intention “C” having been generated first. This is explained in more detail in Chapter 4, where substitution probabilities are estimated by comparing reference transcriptions with recognition output at the intention level.

3.4 Reinforcement learning and Markov decision processes

This section gives a brief overview of the field of *reinforcement learning*, which is concerned with algorithms for finding optimal solutions to *Markov decision processes*. The reader is referred to [92, 93] for more comprehensive surveys.

The problem of interest in reinforcement learning is that of learning from interaction to achieve a goal. For simplicity, this discussion is restricted to interactions of finite length, which are referred to as *episodes*, with tasks in which such interactions take place being referred to as *episodic tasks*. Dialogue is an example of an episodic task, with a single dialogue corresponding to an episode.

3.4.1 Sequential decision processes

A situation where an agent is interacting with its environment by taking successive actions at discrete moments in time, with the environment producing numerical rewards, is referred to as a *sequential decision process* (SDP). It is formally described by defining the following:

State space: Each situation in which the agent has to decide on an action is a state. This is typically defined by the values of a number of relevant variables. The symbol s_t refers to the state at time t (with time $t = 0$

denoting the start of the interaction), while the set of all states (which may be finite or infinite) is the state space $\mathbf{S}$.

Action set: The action set $\mathbf{A}$ is specified by, for each state s_t, specifying the set $\mathbf{A}(s_t)$ of all actions which the agent can perform in state s_t. The action taken at time t is denoted by a_t.

State transition function: The environment in which the agent exists, and the way in which it is affected by the agent's actions, are described by the state transition function. When an action a_t is taken in state s_t at time t, the state transition function specifies the probability $P(s_{t+1}|s_t, s_{t-1}, \cdots, s_0, a_t, a_{t-1}, \cdots, a_0)$ of the state changing to s_{t+1}, given the history of states and actions.

Reward function: After every action a_t the environment produces a numerical reward r_{t+1}. (In cases where all rewards are negative, the term *cost* can be used to refer to the positive value $-r_{t+1}$.) When an action a_t is taken in state s_t at time t, the reward function specifies the probability $P(r_{t+1}|s_t, s_{t-1}, \cdots, s_0, a_t, a_{t-1}, \cdots, a_0)$ of the reward r_t being produced, given the history of states and actions.

The way in which the agent acts is defined by a *policy*, which is a function (denoted by π) mapping the history of states and actions onto the probability of taking each available action:

$$P(a_t|s_t, s_{t-1}, \cdots, s_0, a_{t-1}, \cdots, a_0) = \pi(s_t, s_{t-1}, \cdots, s_0, a_t, a_{t-1}, \cdots, a_0). \quad (3.1)$$

The problem to be solved is to find a policy which maximises the expected long term reward:

$$R = \mathcal{E}\{\sum_{t=0}^{t_{s_f}} r_t\}, \quad (3.2)$$

where $\mathcal{E}\{\cdot\}$ denotes the expected value, and t_{s_f} is the time at which the final state s_f is reached.

If a policy is optimal, it will succeed in maximising R and constitute a solution to the SDP.

3.4.2 Markov decision processes

A Markov decision process (MDP) is a sequential decision process for which both the state transition function and the reward function depend only on the current state and action rather than the entire state and action histories. Thus it has the following two Markovian properties:

$$P(s_{t+1}|s_t, s_{t-1}, \cdots, s_0, a_t, a_{t-1}, \cdots, a_0) = P(s_{t+1}|s_t, a_t). \quad (3.3)$$

$$P(r_{t+1}|s_t, s_{t-1}, \cdots, s_0, a_t, a_{t-1}, \cdots, a_0) = P(r_{t+1}|s_t, a_t). \quad (3.4)$$

When these properties hold, it is possible to find an optimal policy[1] that is independent of the state and action histories, referred to as a *memoryless policy*:

$$P(a_t|s_t) = \pi(s_t, a_t). \tag{3.5}$$

An algorithm that attempts to find an optimal policy of the form of Equation 3.5 is called a *reinforcement learning* algorithm.

3.4.3 State and state-action values

The reinforcement learning problem is made difficult by the fact that the agent does not usually know the state transition and reward functions that define its environment, so that these functions need to be estimated from observations. At the same time, the agent has to deal with the fact that rewards are often delayed: the reward following a "good" action or the penalty following a "bad" action may only occur much later in the episode.

The problem is often approached by estimating the *state value function* $V^\pi(s)$ or, alternatively, the *state-action value function* $Q^\pi(s, a)$. The latter is also referred to as a *Q-function*.

The value of a state s under a policy π is the expected sum of rewards that would be obtained by starting from that state and following the policy π until the end of the episode:

$$V^\pi(s) = \mathcal{E}_\pi \left\{ \sum_{t=0}^{t_{s_f}} r(s_t, a_t) \right\}, \tag{3.6}$$

where s_0 is initialized as s, $\mathcal{E}_\pi\{\cdot\}$ denotes the expected value given that the agent follows policy π, t_{s_f} is the time at which the final state s_f is reached, and $r(s, a)$ denotes the reward received when taking action a in state s.

Similarly, the value of a state-action pair s, a under a policy π is the expected sum of rewards that would be obtained by starting in state s, taking action a, and thereafter following the policy π until the end of the episode:

$$Q^\pi(s, a) = \mathcal{E}\left\{r(s, a)\right\} + \mathcal{E}_\pi \left\{ \sum_{t=1}^{t_{s_f}} r(s_t, a_t) \right\}. \tag{3.7}$$

A policy π_1 is defined to be better than or equal to a policy π_2 if its state value is better than or equal to that of π_2 for all states: $V^{\pi_1}(s) \geq V^{\pi_2}(s)$ for all s. An *optimal policy* is one which is better than or equal to all other policies. At least one optimal policy always exists, and all optimal policies share the same optimal state and state-action value functions, denoted $V^*(s)$ and $Q^*(s, a)$ respectively.

If the function $Q^*(s, a)$ is known, it is easy to follow an optimal policy by, for any state s, always selecting an action a that maximises $Q^*(s, a)$. For this reason, a common approach to solving reinforcement learning problems is to find $Q^*(s, a)$.

[1] While, in general, non-deterministic policies may be optimal, the policies considered in this thesis will be deterministic. An MDP always has at least one deterministic policy which is optimal.

3.4.4 Exploration and exploitation

Estimating $Q^*(s,a)$ is typically done by, after initializing a function $Q(s,a)$ arbitrarily, iteratively updating both the Q-function estimates and the policy for which they are being estimated in such a way that $Q(s,a)$ converges to $Q^*(s,a)$.

In order to achieve this, it is necessary to ensure that the entire state space is explored by trying out all state-action combinations, but also to concentrate on taking the most promising actions, so that the resulting rewards and state transitions reflect what would happen under policies that are close to optimal. This conflict between having to take actions that are relatively unexplored and having to take the actions that, according to the current estimate, are believed to be best leads to a trade-off between *exploration* and *exploitation*.

A simple and popular way to handle this trade-off is to follow a so-called ϵ-greedy policy, where ϵ is a probability $0 < \epsilon \leq 1$. An action that maximises the current estimate of the optimal state-action value function is referred to as a *greedy action*. An ϵ-greedy policy is one that takes the greedy action with probability $(1-\epsilon)$ while exploring by taking a randomly chosen action with probability ϵ. If ϵ is decayed to zero as learning progresses, this approach can be used to guarantee convergence to $Q^*(s,a)$.

3.4.5 Q-learning

Q-learning is a popular method belonging to the class of temporal-difference learning algorithms. The algorithm is based on the following update equation, which specifies the update to be made after action a_t was taken in state s_t at time t, with the observed reward and next state being r_t and s_{t+1} respectively:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha \left[r_t + \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t) \right], \tag{3.8}$$

where α is a variable step size parameter.

This update uses the current estimate of the value of the observed successor state, along with any immediately obtained reward, to refine the estimate of the state-action pair under consideration. The value of the successor state is equal to that of the best (i.e. greedy) action available in that state, whether or not a greedy action will actually be selected. This results in the state-action value function being estimated being the optimal one rather than the one actually being followed, which is referred to as *off-policy* learning. Off-policy learning is useful in that it allows explorative actions to be taken without adverse effects on the estimate of the optimal Q-function.

If the environment is stationary, the Q-learning algorithm is guaranteed to converge to the optimal state-action value function if the following conditions are satisfied:

1. There must be sufficient exploration to ensure that, in the limit, each state-action pair is visited an infinite number of times. This is satisfied by following an ϵ-greedy policy.

2. The step size parameter $\alpha_k(s, a)$ used for the kth update of $Q(s, a)$ should satisfy:

$$\sum_{k=1}^{\infty} \alpha_k(s, a) = \infty \tag{3.9}$$

and

$$\sum_{k=1}^{\infty} \alpha_k^2(s, a) < \infty. \tag{3.10}$$

If the environment is non-stationary, convergence to the optimal function cannot be guaranteed, but it can be tracked by not decreasing the step size to zero.

3.4.6 Monte Carlo learning

An alternative to temporal-difference methods is the class of Monte Carlo learning algorithms. These base the estimate of the state-action value function on the received long term rewards rather than the immediate reward and the estimated value of the successor state.

Monte Carlo algorithms perform the following updates at the end of each episode:

$$Q(s, a) = \mathrm{average}(\mathrm{returns}(s, a)), \tag{3.11}$$

where $\mathrm{returns}(s, a)$ is an accumulated list of the long term reward produced by the state-action pair (s, a). In the off-policy learning case corresponding to the Q-learning algorithm of Section 3.4.5, the list of rewards is accumulated only for instances where the greedy policy was followed from the point where (s, a) occurred until the end of the episode. Convergence to the optimal state-action value function is guaranteed as long as, in the limit, an infinite number of such instances are generated for each state-action pair.

Since Monte Carlo methods perform updates less frequently and at a later stage than Q-learning, they have the disadvantage of often converging more slowly. This is especially the case in the off-policy algorithm mentioned above, where a large number of state-action pairs fail to generate valid returns due to subsequent exploration actions being taken.

3.4.7 Eligibility traces

While Q-learning makes use of estimates of successor state values to enable earlier updates and thus faster learning than Monte Carlo methods, it has the drawback that it may be more vulnerable to violations of the Markov assumptions of Equations 3.3 and 3.4. This is because such violations cause the estimates on which updates are based to be more unreliable than would otherwise be the case.

The use of *eligibility traces* allows the implementation of a continuum of algorithms ranging from Q-learning at one extreme to Monte Carlo learning at the other. In this way, an algorithm can be found that combines the advantages of these two methods.

An eligibility trace $e(s, a)$ is a variable that keeps track of how strong a causal link is assumed between the current state and a particular state-action pair (s, a) from earlier in the episode. Actions taken further in the past are judged to be less directly related to the current state and are therefore less eligible to be updated based on current observations. The eligibility of a particular action is initialized to 1 when the action is taken and then decays exponentially with time. If an exploratory (non-greedy) action is taken later in the episode, the original action is no longer considered a causal factor for the resulting state, so that its eligibility to be updated is set to zero:

$$e_t(s,a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t \\ 0 & \text{if } s = s_t \text{ and } a \neq a_t \\ \lambda e_{t-1}(s,a) & \text{if } s \neq s_t, \end{cases} \tag{3.12}$$

where λ is the eligibility trace decay parameter.

Eligibility traces are incorporated into the update equation as follows:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \delta_t e_t(s,a), \tag{3.13}$$

where

$$\delta_t = r_t + \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t). \tag{3.14}$$

The algorithm described above is known as Watkins' $Q(\lambda)$ algorithm [92, 110]. The nature of the algorithm is determined by the value of λ. For $\lambda = 0$ it reduces to Q-learning, since action values are only updated based on the immediate successor state. For $\lambda = 1$ the algorithm is a version of Monte Carlo learning, which performs updates based on the eventual accumulated reward. Intermediate values of λ produce algorithms that combine the properties of these two approaches.

3.4.8 Dialogue as a Markov decision process

As discussed in Section 3.2.5, a dialogue can be described as a sequence of states, each of which presents the system with a choice of actions from which to select. If, in addition, a reward function is defined that rewards "good" dialogues (e.g. dialogues where the transactions are completed correctly and efficiently), the system becomes an agent in a sequential decision process. The challenge is then to find a state space representation which allows the Markov assumptions of Equations 3.3 and 3.4 to be made, so that an optimal MDP policy can be obtained and used as a dialogue strategy. Finding such a representation using only the information available to the system is not easy, especially since emphasis must be placed on keeping the state space as small as possible. The state space grows exponentially in the number of variables being used to describe it, which means that the learning algorithm can easily become intractable.

It is possible to construct a decision process using the overall information state instead of the system state. Since the system does not have complete knowledge of the overall information state, the process will be only partially observable. In this case, the Markov assumptions lead to what is known as a *partially observable Markov decision process (POMDP)*, for which the exact solution is computationally intractable. However, the fact that the overall information state contains all relevant information may make it less likely that the Markov assumptions will be violated. This is the motivation behind the research efforts described in Section 2.6.2.3 which attempt to find approximate POMDP solutions for this problem. It remains to be seen how such methods would compare with the MDP methods used in this thesis.

The approach taken here is to persist in using the system state, describing the problem as an MDP. If this leads to violations of the Markov property, it has been shown [111] that the learned policies may be suboptimal, with performance arbitrarily far from that of the optimal policy. Also, for temporal-difference learning methods the optimal policy may be unstable, so that even if the problem of local optima is overcome, the algorithm will still fail to converge to the optimal policy. However, for methods such as Monte Carlo learning that use the eventual accumulated reward, the optimal policy for a large class of POMDPs will be stable.

Considerations such as these have led to the proposal that eligibility trace based methods should be used to find policies for processes that may be non-Markov or only partially observable [112]. Empirical results in a number of applications have shown that this alleviates the effect of violations of the Markov assumption, so that policies with good performance are learned. The application of these methods to learning strategies for real systems will be discussed further in Chapter 5.

3.5 Summary

This chapter introduced some concepts that will be useful in the work that follows. Human-machine dialogue was described as an interaction between two agents, which is characterized by the flow of information between them. The structure in this interaction can be described in terms of utterances being exchanged while performing transactions, or in terms of successive dialogue states.

For the purpose of simulating dialogues it is necessary to focus on the user and the communication channel, in order to model the way in which utterances are produced by them. A novel model of utterance generation and corruption was introduced. This will be used to develop a data-driven system for quantitative simulation of dialogue in Chapter 4. This system allows the combination of probabilistic and structural model aspects, a new development in quantitative user modelling.

For the purposes of optimising and automatically designing dialogue strategies, the focus shifts to a state-based description of dialogue. A brief introduction to the field of reinforcement learning was provided, which allows an agent to learn optimal behaviour in a discrete state environment. The application of these methods to the problem of dialogue systems was discussed. Chapter 5 will pursue this topic further, combining the simulation methods of Chapter 4 with reinforcement learning in order to learn strategies for dialogue management.

4

Dialogue simulation

4.1 Introduction

This chapter describes a system that has been developed to simulate cooperative, task-oriented human-machine dialogues, in order to serve as a tool for design and development of dialogue systems. Although the simulation system was developed in the context of specific applications, from which the examples in this chapter will be taken, it should be noted that the system itself is domain independent: as well as describing how the simulation is performed, the chapter aims to illustrate the process of porting to different applications. It should be noted, however, that the systems investigated here focus on information gathering rather than information presentation, where user behaviour may be more unpredictable. Investigation of the extent to which information presentation dialogues are amenable to the simulation approach is beyond the scope of this thesis.

The main focus of the simulations is to enable evaluation of a speech-based dialogue system. For this to be possible, it is necessary to model both the population of users with which the system interacts, and the speech recognition/understanding errors made by the system.

4.1.1 Problem statement

The problem statement for the simulation includes the following:

4.1.1.1 Dialogue system

- A prototype system is available and can interact with humans. In contrast, the simulation approach taken by Lin and Lee [69] does not simulate the performance of a real system, but compares simple handcrafted strategies for an artificial form-filling application.

- The system is deterministic, always reacting to the same situation in the same way. The number of such situations is finite, as is the number of actions that can be taken by the system.

- The system accepts natural speech input.
- The system supports mixed initiative.
- The application is complex enough to require both loops and confirmation sub-dialogues in the dialogue structure.

4.1.1.2 Human user

- The conversation partner for the system is a human user, for whom an “implementation” is not available. This is in contrast to the simulation approach taken by Araki and Doshita [79].
- The user acts in an unpredictable, non-deterministic way, as seen from the viewpoint of the system. That is, the user’s behaviour is too complicated to be modelled deterministically.

4.1.1.3 Communication channel

- The dialogue is conducted via a “noisy” communication channel. In this context, what is regarded as noise includes not only speech recognition errors, but also understanding errors as well as errors on the part of the human user in formulating his/her intentions.

For the purpose of the experiments in this thesis, a prototype cinema information system was used. This system was developed during the latter half of 1998 by a team from Entropic Limited, using a proprietary graphical dialogue design tool. The size of the team varied between 3 and 5 designers, who had previously developed a simple banking application and a few other simple prototypes. The system was developed to replace an existing human operated service and it was intended to be robust both to first time callers and regular users. It was subjected to a small preliminary trial, but the full trial never happened because the team got diverted to another application.

The system provides a telephonic, speech-based interface for a specific application and is implemented using a finite state dialogue structure. A number of allowable syntaxes for speech input are defined, by means of finite state word networks. The word networks are designed so as to support natural language, mixed initiative speech input. A system action consists of producing an output speech prompt and optionally waiting for and parsing speech input, with the parse syntax being chosen from the predefined set according to the dialogue state. It is possible for the same syntax to be used in more than one dialogue state.

The application supports enquiries about cinema information, such as which films are showing at a particular cinema, where and when a particular film is showing, how to contact a particular cinema, etc. The user has almost complete freedom to specify relevant information in different ways, by varying the order, combining different details in the same utterance, or specifying new details in a variety of different dialogue states. The system will be referred to as the *cinema application* and will be used as a source of examples to facilitate the description of the simulation system.

4.1.2 Simulation methodology

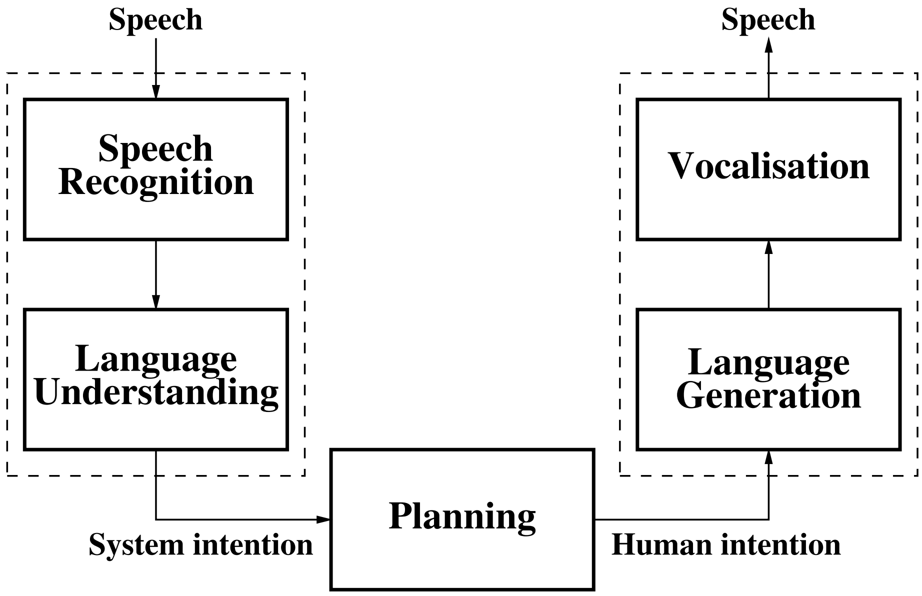


Figure 4.1 *Conceptual block diagram of a human user.*

The first question in simulating the performance of a prototype dialogue system is how the different simulation components should be interfaced with it. For the purpose of answering this question, the user can be thought of as having hypothetical subcomponents mirroring that of an artificial dialogue system, as shown in Figure 4.1.

Next, the assumptions are made that the user can understand the intentions output by the system and that the user's intentions can be mapped onto the intentions understood by the system. The user's recognition and understanding processes are modelled as the inverse of the system's language and speech generation processes, with any deviation to be incorporated into the user model. The required simulation system then reduces to that shown in Figure 4.2.

The planning component in Figure 4.1, along with any distortion (such as user misunderstanding) on the channel from the system to the user, is replaced by the user model component. The transfer function of the channel from the user to the system (user language and speech generation, system speech recognition and system understanding) is replaced by the error generation component. The implementation used here combines the user and error models into a single unit, which can supply the dialogue system with a suitable intention level utterance of the same type as would be output by the speech recognition engine. It is then

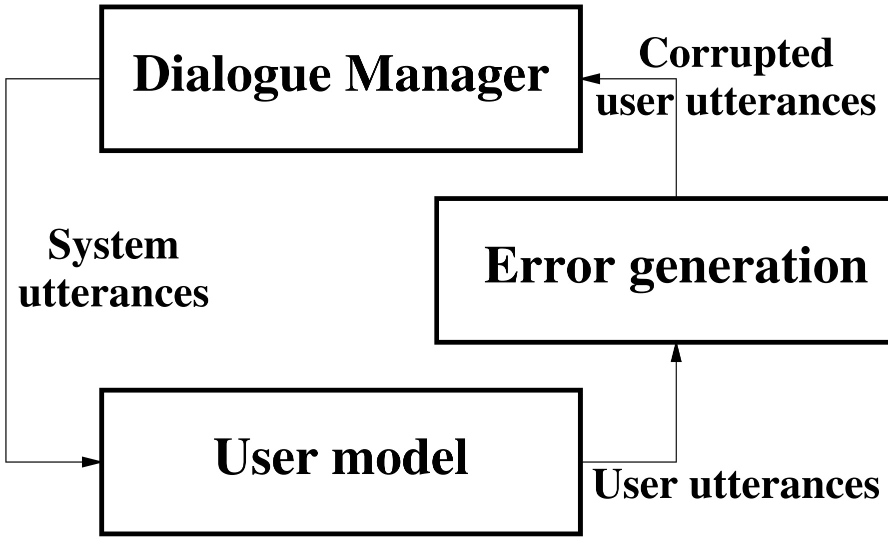


Figure 4.2 *Process of simulating a human-machine dialogue.*

a simple matter to simulate a dialogue by combining the units so that they take turns producing utterances.

4.1.3 Chapter outline

Section 4.2 introduces data-driven approaches to modelling the user. First a simple approach is discussed, in order to motivate the goal directed approach described in the rest of the chapter. The main concepts of the goal directed user model are then presented. A more detailed description is given in the next few sections: Section 4.3 describes the process of constructing the model structure for a given application, introducing and motivating various model components. Section 4.4 details how this structure is to be used during dialogue simulation and Section 4.5 discusses the problem of training the model. Finally, Section 4.6 presents some experimental results according to which the system performance can be judged and conclusions are drawn in Section 4.7.

4.2 Approaches to user modelling

A number of authors [69, 94, 95] have presented simulation experiments based on simple, deterministic user models. These models make assumptions to simplify the human behaviour being modelled, such as that the user always supplies the information being requested without adding anything extra, and always answers confirmation requests correctly. They are useful for artificial problems, but are not used in real applications since they do not attempt to model real user behaviour closely in a quantitative sense. For quantitative modelling of users in real applications, especially applications supporting mixed initiative, a probabilistic approach is required.

4.2.1 Hand-coded versus data-driven approaches

Both non-probabilistic (e.g. [77]) and probabilistic (e.g. [85, 69]) simulation approaches often require the specification of parameters which define the user behaviour. In this way, different styles of user behaviour can be hand-coded and their effects investigated. However, such hand-coded approaches create the problem of finding realistic values for such parameters: since real human behaviour is only poorly understood, it is not obvious which choices of parameters will result in simulations that reflect the behaviour of the user population for which the system is intended.

For this reason, the approach followed here is data-driven: the system learns its parameters automatically from data that has been collected from the target user population. While this has the disadvantage of requiring data collection trials, it allows the construction of a system that reflects the behaviour of real users without the need for the system designer to understand the many subtle nuances affecting it.

For instance, user behaviour may be influenced by aspects such as the wording of system prompts, the recognition performance of the system, limitations in system capability, the level of familiarity the user has with dialogue systems, and so on. These factors are not well understood, and vary between different systems and user populations. It is therefore desirable to measure the user behaviour in each specific case rather than attempting to predict it from general principles. This is done by estimating the required parameters from a data corpus of interactions between users and the specific system being simulated. If a system under development has been changed to such an extent that the associated user behaviour is also likely to change, this approach allows the simulation system to be retrained after collecting a new corpus of data which reflects the way users interact with the new version of the system.

It should be emphasized, however, that data-driven approaches are always dependent on the quality and quantity of the data used to train the models. Since data sparsity is always a limiting factor in these approaches, it is necessary to keep the number of model parameters small. Also, one must bear in mind that the quality of the system can only be as good as that of the data set on which it is trained.

4.2.2 Simple probabilistic user model

In this section a purely probabilistic approach to user modelling is presented, along with some typical output produced by it. It is then argued that a more sophisticated approach is required if realistic simulations are to be produced.

4.2.2.1 Model description

A complete user model should contain a mapping from the full set of states the user can face to the probabilities of the set of allowable user actions. Eckert, Levin and Pieraccini [80, 81] propose a model where the user state is defined as the complete history of both user and system utterances. They then simplify by considering only the single preceding system response, creating a bigram model that specifies the probability of each possible user utterance conditioned on the preceding system utterance. Context other than the immediately preceding response is ignored because of data sparsity. In fact, even in the bigram case it was found that the available data in their application was insufficient so that it was necessary to handcraft some of the parameters.

The approach to be discussed in this section is similar to the one described above. However, rather than estimate probabilities for an utterance as a whole, each utterance is viewed as constructed by traversing a network structure containing a number of choice points, as explained in section 3.3.2. This allows the estimation of separate probabilities for each choice point, which alleviates the data sparsity problem: the average number of occurrences of a single choice point is larger than that of a full path through the network.

The set of allowable user actions is equivalent to the set of intention sequences that can be understood by the system, if this set is extended to include a "mumble" intention for garbage utterances. While for some dialogue systems this set is constant throughout the dialogue, it is often possible to obtain better recognition performance by, in any dialogue state, allowing only those utterances that would make sense in that particular context. For this reason, the prototype system used here links each utterance to a syntax that will be used to parse the user's response.[1] Thus this syntax prescribes a possibly different action set for each user utterance. The action set is described in the user model by a network of choice points, which is constructed from the syntax.

In order to specify the network, it is necessary to specify the list of edges (options) at each node (choice point), with the following information being required for each option:

- The probability with which it is to be chosen. These probabilities can be estimated from data. If a choice point can be used in more than one dialogue state, the probabilities will be state dependent.
- The intention, if any, that is associated with this option.
- The next node (choice point) to be visited in the network. If taking this option completes the utterance, the next node is the terminal node.

In order to generate an utterance using this model, the network is traversed by starting at the initial node and, at each node, selecting an edge according to the

[1] During normal operation, the syntax is used to convert a word level response to an intention sequence. In the simulation, the user model utterances are already at the intention level, so that in this case the syntax acts as a constraint on the set of allowable user utterances.

System:	Which service would you like?
User:	I want to transfer money to my savings account.
System:	How much do you want to transfer?
User:	500 pounds, from my checking account.
System:	Please confirm the following transaction: 500 pounds to be transferred from the checking to the savings account. Is that correct?
User:	I want to transfer from my savings account.
System:	Do you want to transfer from your savings account?
User:	Yes.
System:	Do you want to transfer to your savings account?
User:	Yes.
System:	It is impossible to transfer from your savings to your savings account. Do you want to transfer from your savings account?
Etc.	...

Figure 4.3 *Extract from a typical simulation produced by the simple probabilistic user model. (Intentions have been expanded to the word level in order to construct full sentences.)*

associated probabilities. The utterance is built up by appending the intentions associated with any edges visited. This is continued until the terminal node is reached.

4.2.2.2 Experiment and discussion

The weakness of the model described above is that neither the history of the dialogue (other than the most recent utterance) nor the aims of the user are taken into account. For this reason, phenomena such as inconsistencies between user utterances and aimlessness on the part of the user are to be expected. These expectations were confirmed by trial simulations using handcrafted choice point probabilities. Figure 4.3 presents an extract from a typical result, produced using a prototype banking application. (The intention level utterances have been extended to the word level for readability. This includes specifying a particular value for the cash amount rather than a generic term.) This type of dialogue tends to continue for a very long time, with the user continually changing the transaction specification.

It is clear from the example that the user's actions are unnatural: successive user utterances are inconsistent with one another, impossible goals get specified, and the user's behaviour is generally planless. Another common phenomenon was unnecessary repetition, or re-specification with different values, of transaction details that had already been specified.

An advantage of this model is that there is no need to model errors separately. By estimating the choice point probabilities from the output of the recognition/understanding subsystem, a model is obtained that includes both the user and the error models. Also, some applications may only require dialogues with a simple structure, where each transaction is completed within a few turns and there is no relation between separate user utterances. For such dialogues, this user model would be sufficient.

However, many dialogues require specification of more complicated instructions by the user, where different aspects of the specification are interdependent. This is especially evident in applications where the cost of a system misunderstanding is high enough to require confirmation sub-dialogues, so that the simple probabilistic model is clearly insufficient. This problem needs to be addressed by developing a more sophisticated user model, with the idea of imposing an overall structure on the user's behaviour.

4.2.3 Goal directed user model

The remainder of this chapter presents a more detailed user model that was developed to solve the problems encountered by the simple probabilistic model. The approach is based on the idea of modelling the user state explicitly and in detail, while retaining the probabilistic nature of the model.

In the simple model the only factors determining the user state at the moment when an utterance is generated are the preceding system utterance and the set of options available to the user, as defined by the recognition syntax. The model developed here, however, extends the user state to describe both the user goal and some relevant aspects of the dialogue history. Recall from Section 3.2.3 that a user goal corresponds to a particular transaction which the user wants to be completed. The model is built, therefore, on the central assumption that *the user always attempts to act in accordance with some goal, which remains fixed until it has been completed*. Any deviations from such behaviour are viewed as errors in formulation, which are modelled probabilistically by the error generation module.

By constraining the user actions to be directed towards achieving the user goal and consistent with the dialogue history, a deterministic component is introduced to the user behaviour. It is this deterministic component of the goal directed user model that distinguishes it from the simple model.

The full description of user state used in the model comprises the following:

- The current contents of the user goal
- A set of status variables associated with individual fields inside the goal structure. This enables the model to keep track of facts such as which details have already been specified, which details are believed to have been misunderstood by the system, etc.
- A second goal structure representing the user beliefs on the current system goal: this is updated whenever the system communicates to the user some aspect of what it understands the user goal to be (see Section 4.3.6).

- A third goal structure representing any goal of which the user has just been informed that it has been completed by the system. This allows the user model to verify whether the completed goal was correct and adjust the new user goal accordingly.
- Any necessary application-specific variables describing items such as the type (e.g. direct question) of the previous prompt. Model parameters are conditioned on these variables, so that the simulation designer has some input in choosing contextual variables that may be deemed important in a particular application.

Because of the knowledge-based nature of the goal directed user model, it contains a number of application-specific structures that need to be specified by the designer of the simulation system. The detailed working of the model is best understood by becoming familiar with these structures. To that end, the design and construction of the user model is introduced in the next section, before the actual simulation process is discussed.

4.3 Model construction

In order to introduce the various model components in detail, this section will describe the various steps that need to be performed by the simulation designer when creating the structures to be used for modelling the user and environment for a given application.

It will become clear that, because of the knowledge-based components in the simulation model, this process involves a number of design decisions that require familiarity with the application and desired system behaviour. However, the reward for allowing the designer to make these decisions is that the resulting model acquires flexibility and the ability to model various aspects of real human behaviour that a more simplistic model could not achieve. Also, once these decisions have been specified, further construction of the models occurs automatically.

4.3.1 Intention level language

The dialogue systems that are to be modelled use a set of semantic tags to represent information flowing from the user to the system. Ideally, these will have been designed to be allowable intentions. In practice, however, a dialogue system may use tags that do not conform to the definition[2] of an intention, in which case it is necessary to adapt the tags when defining a suitable intention level language.

For example, a banking application may use semantic tags such as "*savings account*", "*from*", and "*to*". However, none of these can occur independently in an utterance, since the user is not allowed to specify an account name without an

[2]Recall that an intention is defined as the minimum piece of information that can be conveyed independently within a given application.

explicit or implicit specification of whether it is a source or destination account. Similarly, the user cannot convey the concept "*from*" without referring (possibly implicitly) to a specific account. A suitable intention for this example might be "*from the savings account*".

For the purpose of simulating a dialogue, the question of interest is how accurately various intentions tend to get conveyed from the user to the system in particular contexts. For this reason, it is important to work with semantic units as they actually occur in the dialogue. Subdividing intentions into smaller semantic units may be useful for implementational purposes in the dialogue system, but is undesirable for the purpose of modelling, since it fails to make use of the available contextual information. Thus it would either allow modelling of redundant (impossible) contexts, or cause the model to ignore contextual information it could otherwise have taken into account. This is the motivation behind the choice of definition for intentions.

When, as is typically the case for an arbitrary system, the semantic tags do not correspond directly to intentions, the intention level language is defined in terms of the semantic tags used by the system. This is done in two steps:

1. A map is defined from intentions to tags in the system language. This is to be used during simulation to convert utterances (intention sequences) generated by the model into a format that can be processed by the dialogue system.

2. A map is defined from system tags to intentions. This is not used during simulation, but is required for parsing of tag-level training data during model parameter estimation.

Apart from intentions that correspond directly to specific tags or combinations of tags in the application, the following intentions are often useful to define:

Content intentions: Many semantic tags such as "Monday" contain some information (the particular day referred to) which, while necessary to be communicated correctly, is not conceptually relevant to the action of the dialogue system. Thus, while it is important whether or not the content of the tag is recognised correctly, it may not be relevant for the purpose of simulation whether a correctly recognised day is Monday or Tuesday.

For cases such as this, it makes sense to replace the full set of possible tags (a different one for each day of the week) with only two intentions: one for the correct and one for some incorrect value. These are referred to as *content intentions*, to indicate that the intention implies some further information content that is known but not encoded. In the experiments and examples presented here, a naming convention is used whereby intentions indicating the correct value are prefixed by the string "GOOD", while intentions indicating the incorrect value are prefixed by "BAD". Further examples of content intentions may include cash amounts, cinema names, dates, etc. It is the task of the simulation designer to decide whether the specific content of such intentions is relevant enough to the dialogue behaviour under investigation to be encoded explicitly in the intention language, or whether they can be hidden behind a general content intention.

Null intentions: A *null intention* is one that does not map into any tags understood by the system. They are used to indicate the absence of some intention that might have been specified in an utterance. For example, it may be of interest whether, after specifying a cinema name, the user goes on to specify a day in the same utterance. The absence of a day intention in an utterance can be detected and marked by adding a "day-null" intention to the representation of the utterance. Using such null intentions removes the need to model insertion and deletion errors during recognition explicitly, since these just become special cases of substitution errors, where the substitution involves a null intention. It is possible to define various different null intentions to model the different possible omissions in an utterance. In this way, insertion and deletion probabilities for different situations can be treated separately.

Mumble intention: One of the most common types of utterance in human-machine dialogue is (unfortunately) the case where the user says something that is completely unintelligible to the machine, typically because it is not contained in the system vocabulary. This case is represented by the *mumble* intention: it is not the same as silence or other non-speech events, and is useful to model explicitly because of the large number of substitutions involving it. When the user's intended utterance is not expressible in the tag level language of the system, the reference utterance is treated as a mumble, irrespective of how it was interpreted by the recogniser. It is also possible for the intended utterance to be perfectly allowable in the system language, but that the user failed to communicate the meaning to the machine. In this case, the recognised intention will be a mumble, even though the intended (reference) utterance is not. Thus mumbles can occur on either side of the communication channel.

4.3.2 User goal

Goal field	Value	Status
Type:	FILMLIST	specified
Instructions:	NA	-
Town:	NA	-
Film:	NA	NA
Cinema:	GOODCINEMA	urgent
Day:	GOODDAY	pending

Table 4.1 *Example of a user goal and associated status variables in the cinema application.*

The concept of a user goal, as introduced in Section 3.2.3, is central to the simulation approach. The assumption is that the intended actions of the user will be

Goal field	Value
Type:	FILMLIST
Instructions:	NA
Town:	NA
Film:	NA
Cinema:	BADCINEMA
Day:	-

Table 4.2 *Example of a secondary goal structure indicating user beliefs regarding the system goal in the cinema application.*

such as to conform to some predefined goal, which remains more or less constant throughout the course of every transaction. The cases where changes are made to the contents of the goal structure during the course of a transaction will be discussed in Section 4.3.3.

The goal is a simple attribute-value structure consisting of a number of fields, each of which can either be uninstantiated, or take one of a field-specific set of values. Table 4.1 shows an example of a goal structure for the cinema application. In this example, the user is trying to accomplish a "Film listing" goal, to find out which films are showing at a particular cinema on a particular day. The "Instructions", "Town", and "Film" fields are used for other goal types, but are not applicable for this type, while the values of the "Type", "Cinema" and "Day" fields need to be specified. Field values prefixed by "GOOD" or "BAD" denote the correctly and incorrectly recognised versions of content intentions respectively.

The simulation designer must first of all decide how the various tasks that the user might want to attempt can be divided into a sequence of goals, each of which is to be represented by instantiating certain fields. The conditions for successful goal completion can be defined for each goal. The goal structure is created by specifying its fields and the allowable values for each field. Also, certain goal fields may be chosen to be associated with status variables. These variables specify the status of their associated goal fields as one of the following:

Pending: Indicates that the goal field may be specified. This is the typical initial state for a goal field that has to be specified in order to achieve a certain goal.

Specified: Indicates that the goal field has already been specified and should not be re-specified.

Urgent: Indicates that the goal field has high priority to be specified, usually because the dialogue system is believed to have made an error on this field.

Not applicable: Indicates that the goal field is not to be specified at any point in the current transaction.

The dialogue designer has the flexibility to choose which goal fields are to be treated in this way: it is not necessary to track the status of goal fields that may normally be specified repeatedly in the course of a transaction. Goal fields that are not associated with status variables are treated in the same way as those that have "pending" status.

In the example of Table 4.1 the "Instructions" and "Town" fields are not associated with status variables, since in this application they are always prompted for explicitly and in isolation. The other status variables indicate that, at this point in the dialogue, the user has already specified the goal type (and believes the system to have understood it correctly), needs to specify the cinema name with increased urgency (possibly it has been detected from a system prompt that the system has misrecognised the cinema name), and has yet to specify the day.

Table 4.2 shows an example of the secondary goal structure indicating the user's beliefs on the current system goal which accompanies the user goal of Table 4.1. In this structure, the contents of the goal fields are adjusted throughout the course of the dialogue whenever the system communicates new details of what it believes the user goal to be. In the example, the system has communicated that it heard a cinema name. However, this name clashes with that in the actual user goal, so that it is labelled "BADCINEMA" and gives rise to the "urgent" status of the associated variable in Table 4.1.

4.3.3 Effect of dialogue events on the user goal

It is not always the case that the user goal remains constant in real dialogues. It is common, for instance, for users to change their mind on learning more about system capabilities and limitations. Goals also tend to be adapted based on how well the conversation is progressing. Unfortunately, this aspect of user behaviour is very difficult to model and accounts for perhaps the largest deviation of simulations from real dialogues.

The framework presented here provides limited functionality for changing the goal in the course of the dialogue. The simulation designer is required to choose goal and status variable initializations for the various scenarios that are to be simulated, and specify procedures that do the following:

1. Define the initial goal field and status variable settings.
2. Define the new goal field and status variable settings on completion of a goal. This is for simulating scenarios where the user attempts a sequence of transactions in the same conversation.
3. Test whether a goal has been completed correctly. The default test is simply to compare the goal structures representing the user goal and the goal that has actually been completed, but specific goals may require more complex tests, for instance to allow for partial completion of goals.
4. Transform partially valid goal structures into valid goals. This is required for goal inference during training, as discussed in Section 4.5.2.1.

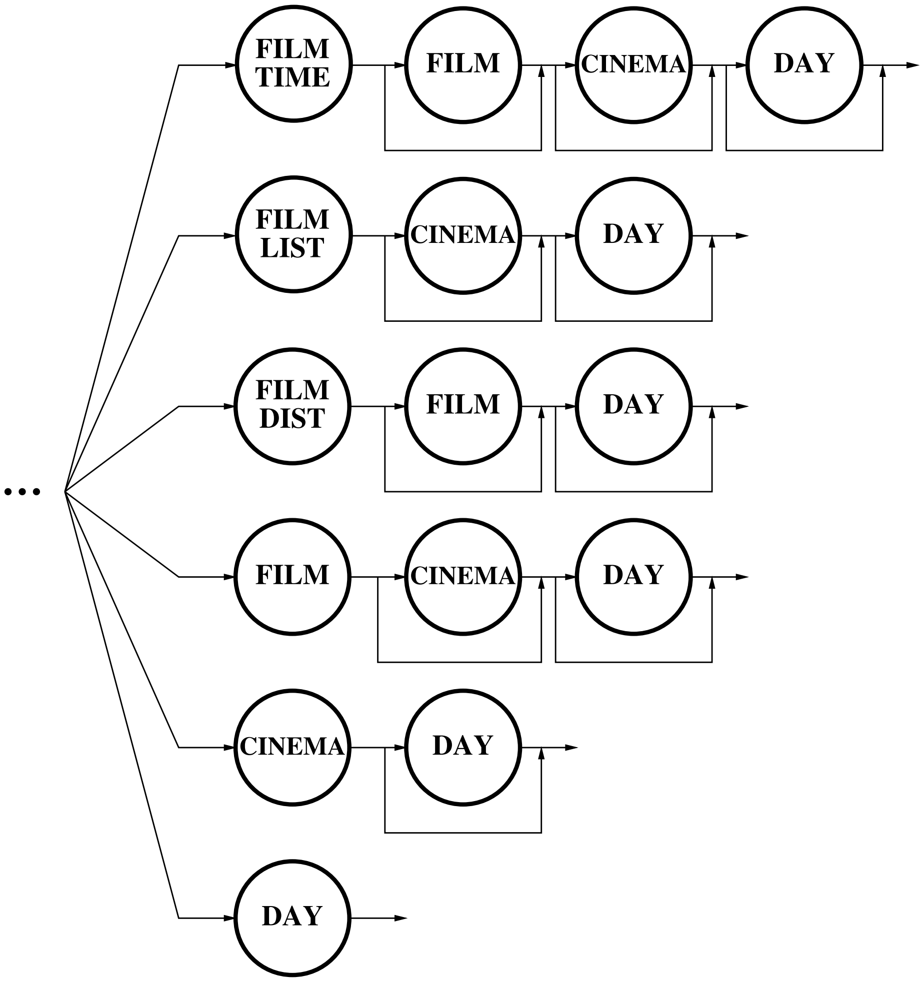


Figure 4.4 *Example of an intention level grammar segment before adaptation.*

5. Define possible direct effects of system actions on the goal. For example, if the system prompts for a detail that has already been marked as specified, the corresponding status variable should be cleared again.

The only goal changes implemented in the system used here are re-initializations on goal completion and simple changes to status variables on detection of system errors. However, if further changes are required for a specific application, they could easily be added by incorporating extra procedures in this section.

4.3.4 Grammar adaptation

The main structural component of the user model is a set of grammars that is used to model utterance generation as discussed in Section 3.3.2.1. These are derived from the grammars used by the recognition engine and correspond closely to intention-level versions of them. Some changes may be required, however, and

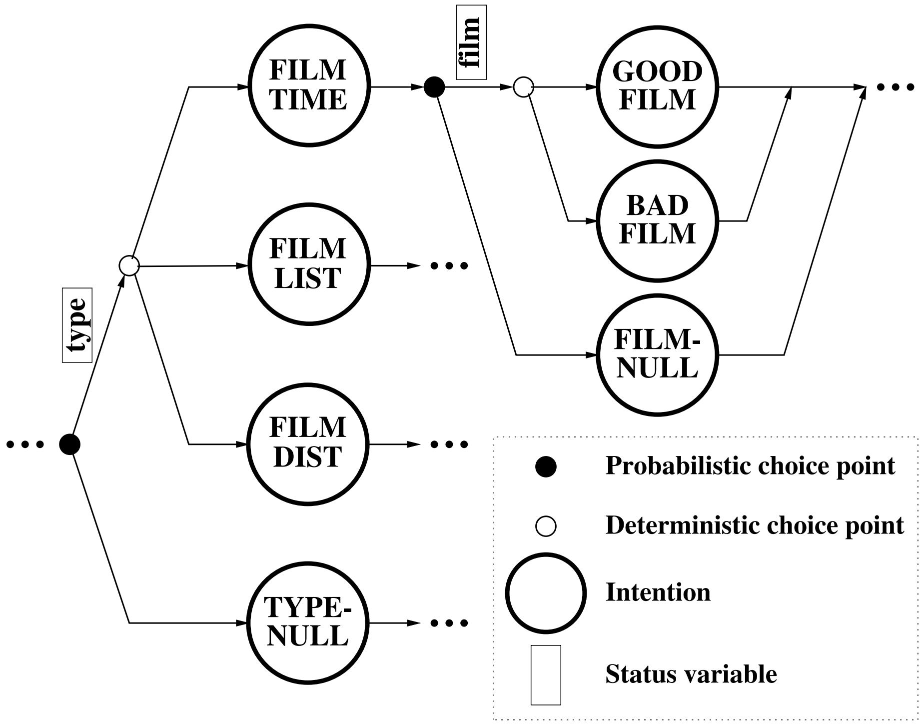


Figure 4.5 *Example of a grammar segment used for utterance generation.*

semantic knowledge has to be incorporated. Figure 4.4 illustrates a segment of an intention-level version of a grammar from the cinema application, while Figure 4.5 illustrates a part of the corresponding grammar used for utterance generation. This grammar might generate the utterance “FILMTIME GOODFILM”, which is an intention level representation of sentences like “Could you tell me when Star Wars is on?” or “I need the times for The Lord of the Rings”.

The steps for creating the grammars are as follows:

1. Simplify the grammars used by the recognition engine to the intention level. Since most of the complexity of the original grammars tend to be at the word level, this step makes the problem of simulating utterances manageable. Each edge in the resulting set of graphs may be linked to an intention, which will usually correspond to one or more of the semantic tags used in the original recognition grammar. After this step, the grammar is in the form of Figure 4.4.

2. Identify nodes that have the same structure (i.e. nodes that give the user the same options) and assign them to backoff groups. In the implementation used here, this is done by means of a naming convention whereby part of the names of nodes belonging to the same backoff group are identical. When parameters are estimated, nodes for which the data is sparse are backed off to parameters estimated for the entire backoff group. For example, the first node in Figure 4.5 is a probabilistic choice between spec-

ifying the transaction type and omitting it for the moment. The same choice also occurs in other grammars (used in different dialogue contexts), so that multiple instances of it can be grouped together. If data for this particular choice point is sparse, the data collected for the entire group can then be used to find out how users resolved the same choice in different dialogue contexts.

3. In general, applications may use grammars that have undesirable properties, so that the grammars for utterance generation must be restructured to eliminate these. In particular, care must be taken to remove loops (to disallow infinitely long utterances) and semantic ambiguity (situations where more than one path through the graph can produce the same sequence of intentions). Ideally, this step should not be necessary since a well-constructed recognition grammar should not display these properties.

4. Add null intentions to mark omissions of particular details. This allows the simulation to model insertions and deletions of those details. Special care must be taken to avoid introducing semantic ambiguity in this step: since null intentions do not appear in the final utterance, ambiguities between paths with different null intentions can easily arise.

5. Distinguish between deterministic and probabilistic choice points. Deterministic choice points are those where the user action depends solely on the current user state, e.g. the goal contents and user beliefs about system state. Probabilistic choice points are those where the user action is to be modelled as a purely probabilistic choice. If there are cases where a choice depends only partly on user state, it should be separated into two choice points, one of which will be deterministic and the other probabilistic.

6. By default, edges associated with intentions are linked to the status variables associated with those intentions.[3] However, in some cases it may be possible that the act of traversing an edge commits the user to specifying the contents of a specific goal field later in the utterance. Since the availability of such edges should depend on the status of the goal field in question, they need to be linked to the associated status variables.

7. For those deterministic choice points (the majority) that can be resolved from the contents of a single goal field, specify the goal field to be consulted and the options to be taken for each possible goal field value.

8. For the remaining (more complex) deterministic choices, specify the name of a procedure to be used to resolve the choice. The content of these procedures will be specified in Section 4.3.5.

[3] In general, an edge may have separate attributes specifying (a) the status variable, or set of status variables, to be consulted to determine the availability of the edge and (b) the status variable, or set of status variables, to be adjusted on traversing that edge along with the values to which they should be set. The implementation described here uses the simplification of allowing only one status variable per edge and forcing it to be set to “specified” status when the edge has been traversed.

4.3.5 Semantic interpretation of user actions

Much of the knowledge-based aspect of the model is incorporated by means of procedures that define how the user is assumed to act in certain specific circumstances. For example, when faced with a confirmation question the natural assumption is that the user will compare the inferred system goal with the real goal, and give a positive or negative confirmation depending on whether or not the compared goals conflict with each other.

In this step, the system designer defines procedures that specify deterministic user behaviour in such circumstances. The procedures may make use of any information included in the user state. Typical examples include the user response to direct (especially yes/no) questions and use of function words specified by the application, such as “cancel”.

In some cases it may be difficult to decide whether to model a particular choice as deterministic or probabilistic. For example, use of the function word “help” could be resolved deterministically by making assumptions regarding typical user actions. However, in such cases where the assumptions to be made are not obvious, it may be better to model the choice as probabilistic instead.

Sometimes even actions such as silence can be deemed to have a specific semantic meaning: silence may, for instance, be interpreted as a positive confirmation after certain indirect confirmation prompts. In such cases, it is up to the simulation designer to decide which assumptions would be realistic.

4.3.6 Information extraction from system output

The output provided by the system typically gives an indication of the system state, and is used by real users to judge the progress of the dialogue. Similarly, it is necessary for the simulated user to use this information to adjust its internal state. The system designer is required to specify, for each relevant system output, what the effect on the user state is. For example, the details specified in confirmation questions and goal completion reports must be extracted and incorporated into the user beliefs on the current system goal and the completed goal respectively. Also, details such as whether the last prompt was a direct question need to be extracted.

4.4 Simulation process

This section provides a walk-through of the process of dialogue simulation, using the structures introduced earlier. Structures for which the exact details have not yet been provided are defined here. It is assumed in this section that the model has already been trained: the estimation of parameters will be discussed in Section 4.5.

4.4.1 Overall simulation structure

A dialogue simulation is performed by running a modified version of the dialogue system software, with the input/output functions overridden by user modelling software. The system output is intercepted by the user model functions of Section 4.3.6 that extract any relevant information from it and update the user state. Calls to the recognition engine are replaced by calls requesting utterance generation by the user model. Calls to possibly unavailable external sources such as databases are also overridden by software that generates suitable imitations of database output.

Thus the system controls the dialogue flow just as if it were interacting with a real user. The user model exists independently, maintaining its own data structures, and is invoked only when the system interacts with it.

4.4.2 Dialogue initialization

At the start of a dialogue, the user goal can be initialized with fixed content in order to simulate a fixed transaction. If the whole range of possible user goals is to be simulated for an overall system evaluation, it is initialized randomly from a list of possible goals, according to the relative frequency with which these goals are observed in data.

The use of content intentions becomes slightly more complicated when the system assumes default values for some variables at the start of the dialogue. For instance, the system might assume that the day of interest is "today", and confirm this assumed value implicitly without explicitly asking the user to specify a day.

This problem can be addressed by setting the map from intentions to system tags at the start of each simulated dialogue to reflect, for each of the assumptions made by the dialogue manager, whether it is correct in the dialogue about to be simulated. This decision is made on the basis of probabilities that have been estimated for each of these assumptions by counting how frequently they are correct in the test data.

After initialization, the system calls upon the user model to generate an utterance once every dialogue turn. The rest of this section discusses what happens during such a call.

4.4.3 Utterance preparation

Before starting to construct the utterance, it is necessary to establish the exact dialogue context. First of all, a check is done to determine whether the system has announced any goal completions since the last user utterance. An announcement of a goal completion may involve the output of some information by the system or a confirmation that some transaction has been carried out. If a goal has been completed, the current user belief about the completed goal, which will have been updated when the system announced the goal completion, is compared with the contents of the user goal structure. If the two goals match, a successful goal completion is logged and the user goal is replaced by the next goal, if any. If there is a conflict, an unsuccessful goal completion is logged. Depending on the application, this situation can be dealt with by forcing the user to try again (the user's belief about the goal completion and the user goal structure are reset) or by continuing to the next task (the user goal is replaced by the next goal).[4]

Next, the utterance preparation software invokes any application specific code (provided by the simulation designer: Section 4.3.6) to adapt the user state according to the exact prompt specified by the system. Once this preparation has been done, control is passed to the graph traversal algorithm, which generates the actual utterance.

4.4.4 Graph traversal

The intention level version of the grammar that would have been used by the recognition engine provides a graph that can be traversed in order to generate an utterance. The traversal is started with a call to process the initial choice point, and continues by processing each successive choice point until the exit node is reached. This section will discuss what happens during the processing of a choice point, which involves the following steps:

1. For probabilistic choice points, the allowable options (edges) are identified by consulting the status variables linked to the options of the choice point.
2. The choice is resolved by selecting one of the options, either deterministically or probabilistically, depending on the type of choice point.
3. Once an option has been selected, it is regarded as the user's chosen intention and any relevant status variables are updated accordingly.
4. If the selected option has an intention connected to it, it is subjected to error generation, during which the effect of recognition/understanding errors is simulated by performing a probabilistic substitution on the selected intention. This may involve jumping to an edge elsewhere in the graph. Also, a confidence measure is generated for the recognised intention.
5. The destination of the edge (possibly changed by the substitution) is identified as the next choice point.

4.4.4.1 Probabilistic choice points

On entering a probabilistic choice point, the options (edges leaving the node) have to be examined to determine which are currently available. This depends on the settings of the status variables. If any of the options are associated with a goal field set as "urgent", a pre-estimated "urgent" probability is used to decide

[4] In the experiments reported in Section 4.6, the latter approach is taken in order to facilitate comparison of goal completion times with those in real conversations, where it is not known whether the completed goals corresponded to the user's actual goals.

probabilistically whether or not to choose an “urgent” option. This probability gives an indication of how often real users specify “urgent” options when faced with such a situation, and will typically be high. In the experiments reported here, it was estimated as a global parameter rather than conditioning it on dialogue context: this helps avoid data sparsity problems.

If the decision indicates that an “urgent” option is to be chosen, the list of allowable options is constructed from all “urgent” options. Otherwise, all “pending” options are used. If there are no urgent or pending options, the status of “specified” options are reset and those are used. If a recognition error has occurred earlier in the utterance, it may also happen that all the options at a choice point are set “not applicable”, implying that the user did not intend to reach this choice point at all. In this case, the choice is made at random between these options.

Each option on the list is associated with a probability obtained from the relative frequencies with which they are chosen by real users. These probabilities are used to select an option as the user’s choice.

4.4.4.2 Deterministic choice points

A deterministic choice point is associated with either a particular goal field or an application-specific function (Section 4.3.5). In the former case, the goal field is consulted and its contents used to indicate the appropriate option. In the latter case, the associated deterministic function is called. These functions have access to the entire user state, and are also able to change the state as a side effect. For instance, they may compare the user goal to the inferred system goal and change goal field status accordingly. The main effect, however, is to select the appropriate option for the circumstance.

4.4.4.3 Updating status variables

If the selected option is linked to a status variable, that variable is set as “specified”, indicating that the user now believes this particular goal field has been specified to the system. This is done irrespective of whether the final utterance passed to the system reflects the chosen option, since the user is not expected to be aware of recognition/understanding errors that may have corrupted the intended meaning.

4.4.4.4 Error generation

If the option that has been selected is associated with an intention (possibly a null intention), a probabilistic substitution has to be performed in order to model possible recognition/understanding errors.

The *context* for this substitution is the point reached in the graph after the last intention was appended to the utterance, or the initial choice point if no intentions have yet been generated. Any intention that could be the next to be generated by starting at this point and following some path in the graph is a candidate for substituting the one that has been chosen. The substitution is done by performing a probabilistic selection among the candidates, using confusion probabilities that have been estimated for each relevant pair of intentions in this context. Usually, the largest probability is for a self-substitution. However, substitutions representing errors, which can also be insertions or deletions when null intentions are involved, also occur.

Once the substitution has been performed, the resulting intention is final: it is appended to the utterance being generated, and further utterance generation continues from the choice point it leads to.

4.4.4.5 Confidence measure generation

The recogniser in the systems being discussed produces a confidence measure for each semantic tag in the recognition output. In order to simulate this, a confidence measure has to be generated for each intention appended to the utterance. These measures are numerical values in the range $[0, 1]$, which, for the purpose of the simulation, are rounded to one decimal place.

Generation of confidence measures is performed using separate confidence measure distributions for correctly and incorrectly recognised intentions respectively. The distributions are specified as sets of discrete probabilities for the rounded confidence measure having each of the possible values $0.0, 0.1, 0.2, \cdots, 1.0$. A confidence measure is generated by selecting the appropriate distribution according to whether the associated intention is being recognised correctly or incorrectly, and then selecting a confidence value according to the distribution probabilities.

4.4.4.6 Example

In the example of Figure 4.5, graph traversal starts at the probabilistic choice point on the left. The first option is associated with the status of the "type" goal field. Assume that this variable is currently set to "pending": in this case all options are allowed and a probabilistic choice is made based on how often users specified the goal type in this situation in the training data. Suppose the first option is taken: the "type" status is set to "specified" and the algorithm proceeds to the next choice point. This is a deterministic choice governed by the value of the "type" field in the goal. If this value is "FILMLIST", the second option is chosen. However, this edge has an intention connected to it which can be misrecognised. Another probabilistic choice is made based on how often, in this context, the "FILMLIST" intention is misrecognised as each of the possible intentions that could have followed the first choice point (including the "TYPE-NULL" intention). Suppose a misrecognition is obtained by generating a substitution of "FILMTIME" for "FILMLIST". The "FILMTIME" intention is appended to the generated utterance and an accompanying confidence measure is generated using the distribution for incorrect recognitions. The algorithm then jumps to the choice point following the "FILMTIME" intention and continues from there until the graph traversal terminates.

4.5 Training process

The problem of estimating the various parameters used during simulation can now be discussed. These parameters are estimated from a data corpus obtained during user trials of an early prototype system. The training data consists of both reference and recognition transcriptions of the user utterances, as well as a record of system utterances. Knowledge of the system state during the dialogues is not required, which helps to make the simulation system application-independent.

It is important, however, to model the fact that the user's actions depend on the particular circumstances s/he is faced with, which include not only the system actions, but also the internal user state. Thus the problem of inferring the user state is central to that of estimating parameters describing user actions.

4.5.1 Data segmentation

The training data are in the form of dialogue transcriptions in the tag-level language of the dialogue system. Both a hand-transcribed reference version and the machine-recognised version, along with the confidence scores generated by the recogniser, are required for each conversation in the data set.

The reference transcriptions are used as an indication of the actual user intentions in the course of the dialogue, and are recognised and parsed by humans according to the grammar used by the recognition engine. While this eliminates recognition and understanding errors, it is impossible to remove communication errors by the user. Thus even the reference data will contain errors such as a user trying to communicate a valid intention using unsupported syntax, or communicating intentions that do not have the desired meaning. In the case of utterances that do not parse, reference transcription consists of the "garbage" intention, which means that the actual user intention is unknown.

The system utterances are analysed in order to subdivide each dialogue transcription into sub-dialogues, where a sub-dialogue represents an attempt of the user to perform a specific transaction, expressible as a user goal. Unfortunately, this can only be done by approximation, by isolating the points where goal completions take place. Since information regarding the real user intentions is not available, cases where the user fails to complete a transaction and proceeds to the next, changing the goal before transaction completion, cannot be distinguished from cases where the conversation started with a simple misunderstanding. However, the assumption made here that user goals do not change in the course of a transaction seems fairly representative of the data.

In the case of failed transactions, such as when the system bails out with a statement like "I'm sorry, I cannot help you", the above assumption is often violated, and such transactions are not used for training.

For the experiments performed here, the exact algorithm for isolating goal completions depends on the particular application. This is because some goal completions are not announced explicitly in the transcriptions, and need to be inferred from other clues available about the dialogue flow. In principle, however, goal completions can be logged while recording data.

4.5.2 User state inference

The user goal, which is assumed to be constant during the course of the transaction, is estimated once for each sub-dialogue, in an initial pass through the data. It is then used while tracking other aspects of user state from utterance to utterance in a second pass through the data.

4.5.2.1 User goal

Since it is not known a priori whether the achieved goals are the correct ones, the user goal must be inferred by looking not at the transaction performed by the system, but at what was requested by the user, whether correctly performed or not. The systems being simulated are required to provide enough feedback for the user to detect errors (whether due to the system or themselves) before completion of a transaction, as well as sufficient opportunity to correct such errors. For such systems, it is reasonable to assume that the correct user goal is defined by the last specification by the user of the contents of each relevant goal field.

Goal inference is thus performed by scanning each user utterance in the sub-dialogue and finding, for each goal field, the most recently specified value for that field. Sometimes a goal field will be unstated, because the system suggests the correct answer using an internal default. In this case the application-specific procedure supplied in Section 4.3.3 is used to complete the partially valid goal.

Apart from being used in the process of estimating the remaining model parameters, the inferred user goal is also used to estimate the relative frequency with which users attempt the various possible goals, which is used to initialize user goals during simulation.

4.5.2.2 Other user state parameters

The remaining user state variables change throughout the course of the transaction, and must be kept updated as the dialogue progresses during parameter estimation. The necessary steps include the following:

1. Extract information from system prompts by running the procedures supplied by the simulation designer in Section 4.3.6. User state variables that can be affected include those that indicate specific prompt types, the assumed system goal and the assumed completed goal.

2. Perform the same pre-utterance processing that is done during simulation. This may also affect the goal status variables.

3. Set the status variables while processing the utterance in the same way as during simulation. This includes running any application specific procedures that would have been run during simulation, since these procedures may have the ability to change the user state.

To take account of the continuity of the user state in real dialogues, the state is reset only at the start of a new dialogue and not between successive transactions in a dialogue. This has the effect of retaining contextual information so that the user can refer to something stated in an earlier transaction of the same dialogue without explicitly stating it again.

4.5.3 Parameter estimation

The parameters for the model are in the form of probabilities or frequency counts from which probabilities can be obtained. In all cases, probabilities are obtained using simple maximum likelihood:

$$P(Event|Context) = \frac{cnt(Event, Context)}{cnt(Context)}, \tag{4.1}$$

where $cnt(Event, Context)$ is the counted number of occurrences of the event in question in a specific context and $cnt(Context)$ is the counted number of occurrences of the context irrespective of whether or not the event occurred. Counts are obtained for both the exact context and a wider backoff context, so that the backoff context can be used where data sparsity occurs.

In order to establish which parameters are to be updated, both the reference and recognised versions of each training utterance must be parsed so that the exact sequence of choice points is known. The path followed by the reference utterance is to be used to estimate parameters for the user model, while the error model is obtained from a comparison of the reference and recognised versions.

The parsing is done by performing a complete search to find all possible paths through the utterance generation graph. If more than one path is possible, the path with the largest number of non-null intentions is the correct one.

4.5.3.1 Modelling user actions

Once the correct paths have been found, the graph is traversed along the reference path, and the following updating steps are performed at each node:

1. Find out which options were marked by status variables as “pending” and “urgent”.
2. If there was an “urgent” option, update the counts for the urgent probability according to whether or not the chosen option was urgent.
3. Identify the context as the specific node in the graph, with the backoff context being the set of all nodes in the same backoff group.
4. If the chosen option was either “pending” or “urgent”, and at least one alternative was open to the user (i.e. an unchosen option marked “urgent” or “pending”), increment the context occurrence count for all available options and the event occurrence count for the chosen option.

Note that if the chosen option was not "urgent" or "pending", the user action was not goal directed. No parameters are updated in this case. Also, automatic choices where the user had no valid alternative are not counted and context occurrence counts are not incremented for options that had goal status settings making them unavailable.

After obtaining the various counts in this process, Equation 4.1 can be used to calculate the probabilistic choice point parameters that govern user behaviour in mixed initiative situations, the "urgent" probability that governs user behaviour after misunderstandings, the frequencies with which different goals are attempted and the probabilities of default system assumptions being correct.

Since not all the options at a choice point are necessarily available every time it is visited, it is sensible to store the counts rather than the probabilities. During simulation, the probabilities can then be calculated based on the counts for the available options only.

4.5.3.2 Error modelling

In addition, the following updating steps are performed at each node where an intention was produced in the reference utterance:

1. Find the identical context in the recognition path through the graph. Updating is only done if such a context exists.

2. Content intentions are treated by using separate intentions for correct and incorrect recognition. The reference intention is always assumed to be correct. If the recognised intention is a content intention that is different from the reference intention, it is assumed to be the incorrect version. If both reference and recognition correspond to the same content intention, the utterances are compared to decide whether the recognised intention is the correct or incorrect version.

3. Identify the context choice point as the point in the graph immediately after the last intention was produced, or the initial choice point in the graph if no intentions have yet been produced. The full context consists of this context choice point and the reference intention. The backoff context is simply the reference intention, with the point in the graph being ignored.

4. Increment the context occurrence count for all intentions (either null or non-null) that could be produced as the first intention after the context choice point.

5. Increment the event occurrence count for the intention that was actually recognised.

When the entire data set has been processed in this way, the necessary counts are available so that, during simulation, relative probabilities can be computed for the options that are available at any particular time. Counts have been obtained for both specific and more general contexts, so that it is possible to back off to the general case whenever the number of training examples falls below a preset threshold.

4.5.3.3 Confidence measure parameters

The distribution of confidence measure values naturally depends largely on whether the associated intention was recognised correctly or incorrectly. Due to data sparsity, it is necessary to ignore other factors such as dialogue context, that might also potentially influence the distribution.

Confidence measure distributions for correct and incorrect recognition are estimated by dividing the data into these two classes and rounding the confidence measures to one decimal place. The probability of each bin in $\{0.0, 0.1, 0.2, \cdots, 1.0\}$ is then calculated from its number of occurrences in the data using Equation 4.1.

4.6 Experiments

4.6.1 Experimental setup

4.6.1.1 Goal of experiment

The purpose of the experiments reported here is to ascertain to what extent the simulated dialogues give an accurate picture of the dialogues produced by real users using the system in the task domain.

This was done by using an existing application with which a test corpus of real dialogues had been built up. The corpus was divided into training data, used to train the simulation system, and test data with which the simulation results are to be compared. A number of scenarios that occur in the corpus were selected to be simulated.

Unfortunately, comparing the simulation results to real dialogues is problematic due to the difficulty of analysing the real dialogues. Whereas, for simulated dialogues, information on what the user is trying to accomplish is always available, the user goals in spontaneous real dialogues are unknown. Thus it is impossible to ascertain which of the achieved goals in real dialogues are in fact what the users intended and which are the results of misunderstandings.[5] For this reason, goal achievement rates calculated for the real data cannot be expected to be accurate. Further, while average transaction completion times can be calculated, it is not clear to what extent the calculated figures are affected by the fact that the data may contain both correct and incorrect goal completions. Another important factor is that many of the transactions are initiated at some stage in the course of a long dialogue, so that the exact dialogue context is different for different occurrences of the same transaction. This has a large effect on the time taken to complete the transaction: if, for example, the user asks for the times of

[5] While this problem can be addressed by asking callers to act out specific scenarios, this approach has also been found to present problems, such as the difficulty in communicating the exact scenario task to the caller and the fact that callers are not motivated as they would be if they were using the system on their own initiative.[33]

a particular film immediately after asking for a list of films showing at a certain cinema, the cinema name remains in context so that time is saved by it not having to be communicated a second time. This causes average completion times measured on the test data to become less meaningful.

In spite of this, goal achievement rates and goal completion times were calculated for the simulated dialogues, and are compared to the corresponding statistics from the test data to give an indication of the performance of the simulation.

4.6.1.2 Test application

The experiments reported here were performed using the cinema application mentioned in Section 4.1.1. The application allows the user a large amount of freedom in the way s/he can set about requesting information. For instance, a large range of syntactical constructs can be used, with the result that intentions can be strung together almost arbitrarily in an utterance. The syntax also allows loops, so that the same intention (or different versions of the same content intention) can be specified an unlimited number of times in the utterance.[6] Also, the same syntax is used in various different dialogue states, so that the user has a full range of options available in most of the possible states. This highlights the importance both of using a goal directed approach in modelling the user and of tracking the dialogue progress in the user state representation: the user actions depend on factors unrelated to either the syntax or dialogue state with which the user is presented, and these factors change through the course of the dialogue.

The application starts off with a structured section of dialogue, during which the user is offered an opportunity to receive instructions on how to use the system, and the town for which information is sought is established. The system was designed with the limitation that information can only be obtained about one town per dialogue. After this preliminary section, a general state is reached where the user is allowed to make a series of enquiries about film names, film showing times, cinema names and cinema addresses.

The goal structure used for this particular application contained the following fields:

- Instructions: A field used in the preliminary section of the dialogue, taking the values "YES" or "NO", used to indicate whether the user wants instructions on system usage if offered them. Since separate data for expert and non-expert users was not available, the same model was used for both and the setting of this goal field had no effect on other parts of the dialogue.

- Town: Another preliminary field, taking the name of a system-recognised town as its value. Since the town is treated as a content intention, the

[6] Since re-specification of the same detail does not make semantic sense, however, only the last specification will be used in the semantic processing. This allows a simplification of the intention level syntax to avoid loops. In general, it will always be possible to avoid looping topologies in the intention level syntax, since, by definition, multiple occurrence of an intention in an utterance is meaningless.

value used here is simply the intention indicating a correctly recognised town (denoted “GOODTOWN”).

- Type: Indicates the type of enquiry. The value of this field is taken from “FTIME”, “FLIST”, “FDIST”, and “CADDR”, indicating enquiries concerning film times, film listings, film distributions and cinema addresses respectively.
- Film: A content intention indicating the name of the film being enquired about. The value for a correctly recognised film is denoted “GOODFILM”.
- Cinema: A content intention indicating the name of the cinema being enquired about. The value for a correctly recognised cinema name is denoted “GOODCINEMA”.
- Day: A content intention indicating the day being enquired about. The value for a correctly recognised day is denoted “GOODDAY”. The system uses different semantic tags for days specified by name and days specified relative to the current day (e.g. “TODAY” or “TOMORROW”), but these differences were suppressed at the intention level.

Using this goal structure, the following possible user goals were defined according to the functionality offered to the user:

- Instructions: Obtain instructions for using the system on entering the dialogue.
- Town: Specify the town about which information is desired.
- Film times: Find out at what times a specific film is being shown at a specific cinema on a specific day. This involves specifying the goal contents for Film, Cinema and Day.
- Film listing: Find out which films are being shown at a specific cinema on a specific day. This involves specifying the goal contents for Cinema and Day, while the Film field is uninstantiated.
- Film distribution: Find out at which cinemas a specific film is being shown on a specific day. This involves specifying the goal contents for Film and Day, while the Cinema field is uninstantiated.
- Cinema address: Find out the address and booking number for a specific cinema. This involves specifying the goal contents for Cinema, while the Film and Day fields are uninstantiated.
- Exit: Close the dialogue by eliciting the exit message from the system. This involves communicating that the user has no further goals. (All goal fields are uninstantiated). The exit goal fails if the user fails to communicate this and is forced to hang up without concluding the conversation politely.

With the above goals being defined, they can be combined in sequences to create specific scenarios to be simulated. Unfortunately, however, the corresponding scenarios for real users in the test data are unknown, and vary from case to case. Thus we are forced to rely on the assumption that the transactions performed in the context of the simulated scenarios are representative of the way the same transactions will be performed in the entire set of possible scenarios.

The choice of scenario can have a large effect on goal completion times, and it is not always clear which scenarios should be chosen. For instance, the cinema address goal will usually only be attempted after the desired cinema name has been mentioned by either the user or the system while pursuing one of the other main goal types. This means that the cinema name need not be respecified by the user, which shortens the goal completion time. In the experiments reported here, this goal was attempted after a film listing goal. Other goals were attempted in isolation, except for the film times goal, for which we defined two variants: "Film times (1)" attempts the goal in isolation, while "Film times (2)" attempts it after successful conclusion of a related "Film Listing" goal, the details of which remain in the memory of the dialogue system. The same test data was used for both variants.

4.6.1.3 Data corpus

The data used for these experiments was collected by recording trial runs of the cinema information system, with real users calling in to use the system. The corpus consists of 911 dialogues with a total of 7409 utterances. Of these dialogues, 134 were failed conversations where the user did not accomplish any goals. The number of instances where a transaction of one of the main goal types ("FTIME", "FLIST", "FDIST" and "CADDR") is completed, is 466.

Because of the high variance of the various goal completion times over the corpus, it is undesirable to use a small proportion of the data as a test set. To overcome this problem, the data was partitioned into 10 sections, each of which was used as a test set while the remaining 9 were used for training. The results over all 10 such runs are combined to obtain a better picture of system performance.

4.6.1.4 Discussion

There are a number of reasons why the results of the comparisons made in these experiments can only be expected to give a rough indication of the system performance. The most important of these is the low level of reliability of the figures obtained by analysis of the real data. This analysis assumes, for instance, that goal completion times are independent of whether the goal is achieved correctly or not. It also neglects the effect of goals being attempted in different dialogue contexts, while the proper contexts to be used for comparative purposes in simulations are unknown.

Estimation of transaction failure rates is particularly problematic, since the true user goals are unknown. Users also often tend to change their goals when they become aware of system limitations that make accomplishment of their original goal seem more difficult.

Since the corpus is limited in size, it is also very sensitive to outliers in the data caused by individual callers who may be experimenting with or testing the system rather than using it in the normal fashion. The measured goal completion times should therefore not be expected to give more than a relative indication of how the times for different goal types compare with each other.

The simulation system, on the other hand, is designed to produce dialogues that reflect the average behavioural patterns of the training population as a whole. This can be expected to result in a simulated data set that is smoother, since all the dialogues are produced by the same user model rather than different individuals. While this smoothing effect may make the simulated dialogues unrealistic in the sense that their goal completion times and achievement rates do not display the same variance as those of real dialogues, the averaged statistics should not suffer, and could even be more accurate than those measured on a small amount of real data. Also, smoothing the statistics of the training population buys robustness against outliers in the data.

4.6.2 Simulation results and conclusions

The simulation system implemented for the cinema application has 135 probabilistic choice points and 653 substitution contexts for which a total of 326 choice probabilities and 3069 substitution probabilities, many of them backed off to more general cases, are estimated respectively. The system was trained on each of the training sets, and used to simulate 1000 dialogues.

Figures 4.6 and 4.7 show typical simulations, produced using the same user goal. In the first dialogue the user takes no initiative, while in the second the mixed initiative ability of the system is utilised. Note also how the first dialogue results in a failed transaction when the user responds with silence to a confirmation prompt. This happens because, in the training data, users were often silent in response to this particular prompt even when the detail being confirmed is wrong.

Goal	**Simulated failure rate (%)**	Test failure rate (%)
Film Times (1)	15.88	9.76
Film Times (2)	7.62	9.76
Film Listing	7.85	3.51
Film Distribution	2.82	1.49
Cinema Address	9.08	4.13
Weighted Average	6.89	5.26

Table 4.3 *Simulation results: average transaction failure rates for different goal types in simulated and test dialogues.*

Figure 4.8 shows, for the various goal types, the average goal completion times found for both the simulation system and the test data averaged over all the data partitions. The error bars indicate the 95% confidence intervals for the measured values. Transaction failure rates for simulated and test data measured for the

Dialogue 1:	
System:	Welcome to the Talking Pages cinema information line. Would you like instructions on how to use this system?
User:	No.
System:	Which town or city would you like cinema information for?
User:	[Unparseable utterance, transcribed as a mumble.]
System:	I'm sorry I didn't understand that. You can choose from Bristol, Reading, Basingstoke, Wokingham or Slough. Please say the name of the town or city you would like.
User:	Bristol [Misrecognised, with a confidence score of 0.6.]
System:	Cinema information for Slough. [Confirmation.]
User:	I said Bristol.
System:	Cinema information for Bristol.
User:	[Silence.]
System:	Remember to say Help if you are unsure of how to respond. What information would you like?
User:	Film times
System:	Please confirm or correct me. Do you want times for a film today?
User:	Yes.
System:	Which film would you like to see?
User:	Eyes Wide Shut.
System:	At which cinema would you like to see "Eyes Wide Shut"?
User:	The Odeon. [Misrecognised, with a confidence score of 0.3.]
System:	The ABC. [Confirmation.]
User:	[Silence, incorrectly interpreted as positive confirmation.]
System:	It is on at... Can I help you with anything else?
User:	No thanks. [The output was for the wrong cinema, meaning a failed transaction. For this experiment the user was hard-coded not to attempt to repeat failed transactions.]
System:	Thank you for calling the Talking Pages cinema information line, goodbye.

Figure 4.6 *Typical simulation produced by the goal directed user model: Example 1. Intentions have been expanded to the word level in order to construct full sentences. Comments are added in square brackets.*

Dialogue 2:	
System:	Welcome to the Talking Pages cinema information line. Would you like instructions on how to use this system?
User:	No.
System:	Which town or city would you like cinema information for?
User:	Bristol.
System:	Cinema information for Bristol.
User:	[Silence.]
System:	Remember to say Help if you are unsure of how to respond. What information would you like?
User:	Eyes Wide Shut at the Odeon.
System:	Please confirm or correct me. Do you want times for "Eyes Wide Shut" at the Odeon today?
User:	Yes.
System:	It is on at... Can I help you with anything else?
User:	No thanks.
System:	Thank you for calling the Talking Pages cinema information line, goodbye.

Figure 4.7 *Typical simulations produced by the goal directed user model: Example 2. Intentions have been expanded to the word level in order to construct full sentences.*

main goal types are shown in Table 4.3. The average shown is for the last four entries, weighted by their respective goal frequencies. Unfortunately, the number of test dialogues in which these goals were completed is too low for the failure rates obtained for the individual goal types to be meaningful. The average failure rate shown for the test data is also unreliable due to the uncertainties involved in estimating the true user goals.

From these results it is clear that there is good agreement between the simulations and the test data, although the predictions made by the simulations are of course not exact. The agreement observed for the simple goals is very good: each of "Instructions", "Town" and "Exit" involve communicating the contents of only one goal field to the system (for the "Town" goal a confirmation is done, which accounts for the extra transaction length). The four main goal types are more complex, involving communication of various combinations of goal fields, with the time taken to complete the goal in both the simulations and the real data roughly correlating with the amount of information that needs to be communicated. The transaction type is required for all of the main goal types, but can sometimes be inferred by the system without being explicitly stated by the user. Apart from this field, "Film Times" requires specification of a film name, a cinema name and a day, "Film Listing" requires specification of a cinema name and a day, "Film Distribution" requires specification of a film name and a day, and "Cinema Address" requires specification of a cinema name only. "Film Distribution" is slightly faster and has a much lower failure rate than "Film Listing"

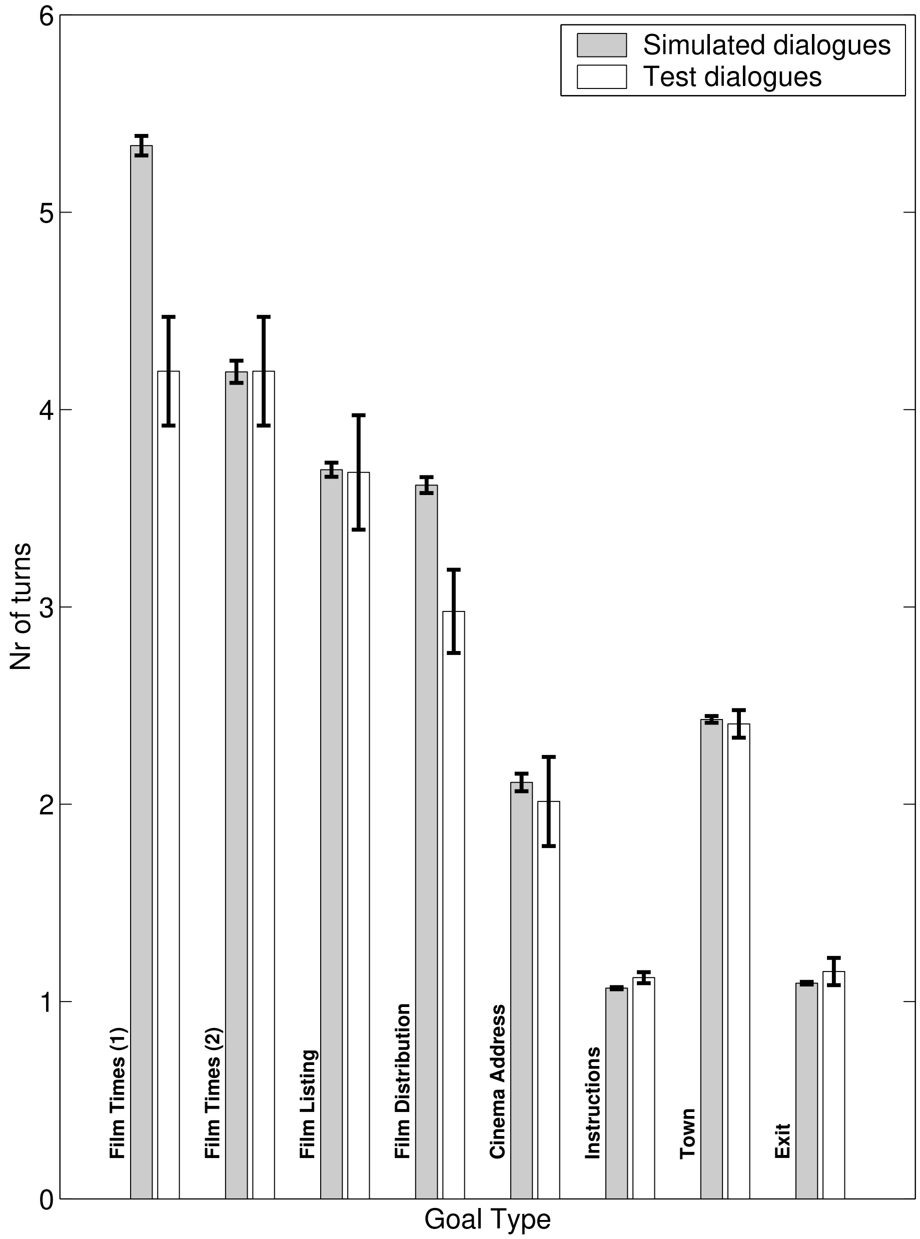


Figure 4.8 *Simulation results: average goal completion times (number of turns) for different goal types in simulated and test dialogues.*

because film names have a higher recognition accuracy than cinema names in this application.

The main goals prove difficult to simulate accurately, with a major factor being the effect of the different contexts in which the goals are attempted. The effect of varying this context is illustrated by the difference in performance for the two "Film Times" goals: the version where an arbitrary context is assumed ("Film times (2)") agrees substantially better with the test data than the initial version where no context is used. This is presumably also the reason for the discrepancy between predicted and observed completion times for the "Film Distribution" goal: it was simulated with no context, while in some of the real dialogues the name of the film may already have been in context. In general, the effect of discarding inter-goal context in this application is to increase the predicted goal achievement time by forcing the maximum number of goal fields to be communicated.

It is important to note that, in spite of the discrepancies between the real and simulated dialogues, the simulations do manage to recognise the relative differences between the different goal complexities and thus predict their relative completion times. For the purpose of relative evaluation of dialogue strategies, the simulations should thus be sufficient even if it is not advisable to rely on the prediction of dialogue duration as an absolute measure.

4.7 Summary

The problem addressed in this chapter is the development of a simulation system for human-machine dialogues, and in particular the modelling of user actions and speech recognition performance. Previous attempts to model the user used much simpler approaches, which prove insufficient for the applications of interest here. Also, previous attempts at simulating dialogue have either neglected errors completely, or used a simplistic method for simulating them.

This chapter presented a novel approach to the problem of user modelling, based on the assumption that the user acts in a consistent and goal directed fashion, and a detailed model of the internal user state. The structure provided by this model is combined with a probabilistic model of both user actions and recognition and understanding errors, leading to a system that is capable of producing realistic looking simulations of dialogues with complex structure. The performance of the simulation system was evaluated by means of comparative experiments on real data, which demonstrated that it simulates different transactions well enough to perform relative predictions of their durations.

5

Strategy design using reinforcement learning

5.1 Introduction

Section 2.6.2 introduced two strategy learning approaches based on Markov decision processes. The model-based approach has been shown to be usable for real applications, while the simulation-based approach adopted in this thesis has previously only been applied to very simple systems, due to the lack of an accurate dialogue simulation system. In this section the limitations of these approaches are discussed and their respective advantages and disadvantages are compared.

5.1.1 Limitations of reinforcement learning approaches

In both the model-based and simulation-based approaches, the eventual system performance is limited by the following factors:

Data quality: The data used to train the models is collected in preliminary user trials, the nature of which can have a crucial effect on the accuracy of the model. Relevant factors include:

- whether the users are representative of the real user population, especially in terms of level of expertise with dialogue systems
- whether the users are motivated to complete transactions as real users would be when using the system, or acting out scenarios of which the correct completion is not necessarily important to them
- whether the data set is large enough

Model assumptions: Crucial assumptions include the assumption that the chosen state space representation is Markov and that the user and environment statistics are stationary.

Evaluation function: The learning algorithm optimises the strategy with respect to some evaluation function. It is important for the choice of this function to be appropriate, so that optimising it is as close as possible to optimising the qualitative system performance.

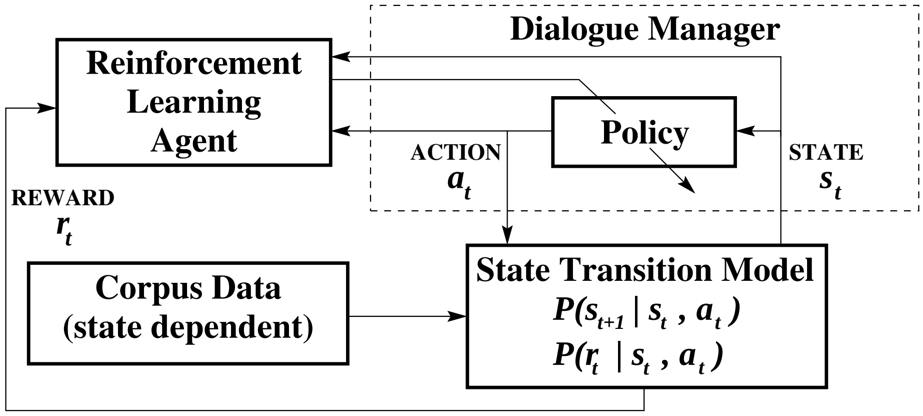


Figure 5.1 *Block diagram of the model-based strategy learning approach.*

5.1.2 Comparison of model-based and simulation-based approaches

Figure 5.1 illustrates the model-based strategy learning approach. The dialogue strategy is determined by the policy (mapping from states s_t to actions a_t), which is controlled by the reinforcement learning agent. The policy is interfaced directly with a state transition model: this forms the environment with which the agent is interacting. The state transition model is trained from a data corpus that has been transcribed at the state level. It consists of estimates of the probabilities $P(s_{t+1}|s_t, a_t)$ and $P(r_t|s_t, a_t)$ for all state-action pairs (s_t, a_t), which govern the transition from state s_t to state s_{t+1} and the observed reward r_t when action a_t is taken at time t. During strategy learning, the agent uses a reinforcement learning algorithm to find a policy that maximises the expected long term reward.

In order to obtain a good state transition model, the data set should contain dialogues in which all system state-action combinations are represented. The dialogue state transitions have to be logged, ideally during data collection, since the required data set consists of the state sequences observed in the course of the trial dialogues. While it is possible to reverse engineer a log of these transitions if the state space representation is changed, a new representation may result in the data no longer representing the full set of state-action combinations. This means that, ideally, the system state representation should be fixed at an early stage. The chief advantages of this approach are that no assumptions need to be made about the users, other than that they give rise to a state transition process which is Markov and stationary, and that the state transitions in the data set are known exactly.

The simulation-based approach is illustrated in Figure 5.2. As in the model-based approach, a policy is being controlled by a reinforcement learning agent, using the observed states, actions and rewards. Instead of interfacing with a state transition model, however, the policy interfaces with a user and error model of the type developed in Chapter 4. The interface is managed via a domain model, which performs the translations from actions produced by the policy to intentions understood by the user model and from user intentions to system states. The user

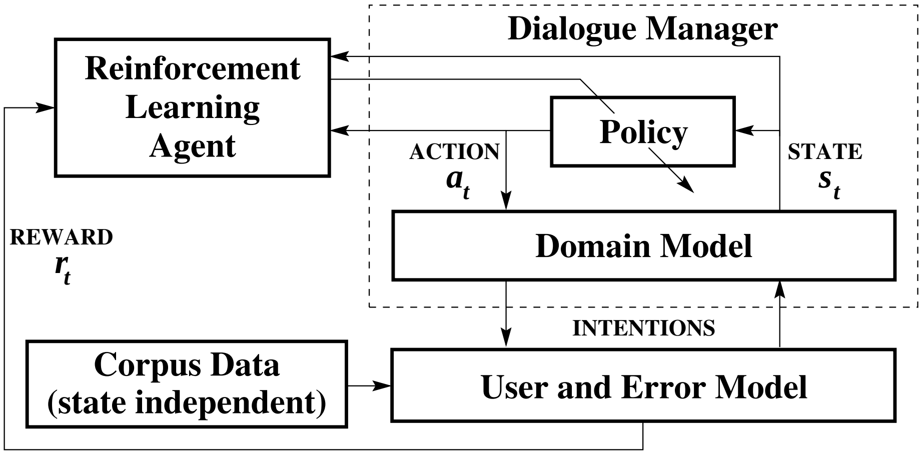


Figure 5.2 *Block diagram of the simulation-based strategy learning approach.*

and error model is trained from a data corpus that has been transcribed at the intention level.

In the case of this approach, the data set should be representative of all user states, but need not be representative of all system state-action combinations. This leads to the availability of a natural back-off mechanism for estimating parameters in the user model to alleviate the data sparsity problem.

While the user state representation is fixed before these states are estimated from data, the system state representation need not be fixed. Thus if the chosen system state representation proves problematic during training, it can be changed and the system retrained without having to change the data set, data transcriptions or user model. This flexibility, which allows the utilisation of fixed data corpora, transcribed without reference to system state, is the main motivation for adopting the simulation-based approach.

A further potential advantage of this approach is the possibility of bootstrapping the user and recognition error models from general user models and separate recognition statistics respectively. The recognition syntaxes proposed for use by the system can be tested individually and the results applied directly rather than having to rely solely on recognition results observed in a dialogue context, which is likely to be more sparse. Although the reliability of this approach has not yet been investigated, it would allow great savings in the amount of application-specific data required.

The extra detail obtained by modelling at the utterance rather than the state level can also be useful for qualitative evaluation of the model and of the trained system by humans, since it can be used to generate typical intention level dialogues rather than just state transition sequences. It should, however, not make any difference to the quantitative evaluations performed by the learning algorithm.

The disadvantage of this approach is that explicit assumptions are made about users, both in order to estimate their assumed state and to model their behaviour.

Naturally, the approach is only valid to the extent that the model assumptions are justified. Also, the accuracy of the simulation is of course limited by the quality and quantity of the data just as in the model-based approach. While the use of back-off techniques alleviates the effect of data sparsity to some extent, one cannot expect accurate results with either approach if the data set is too small.

5.1.3 Chapter outline

The learning approach used here is described in more detail in Section 5.2. The remainder of the chapter then discusses the application of this approach to two specific problems. In Section 5.3 an abstract representation is defined for an artificial form-filling application and used to perform preliminary experiments investigating the proposed approach. Basic parameter settings are investigated and convergence of the algorithm to the correct policies is confirmed in simple cases where the optimal strategy is known. The ability of the algorithm to learn different policies for different user and error models is also confirmed. More complicated cases are then explored in order to verify the ability of the algorithm to learn good policies when the optimal solution is not known in advance. In addition, these experiments demonstrate that the learned policies are sometimes nonuniform in their treatment of different slots, which is not usually the case in manually designed systems.

Finally, Section 5.4 applies the approach to a real dialogue application. The scale of these experiments is much larger than for the artificial problem, so that convergence of the learning algorithm becomes more problematic and the performance of different learning algorithms can be compared. Strategies are learned for different evaluation functions and evaluated by comparing them to the original manually designed system. The learned strategies are also analysed in order to see in which way they adapt to optimise the different criteria.

5.2 Description of simulation-based approach

The initial stages of the dialogue design process include choosing the action set for the system, designing a recognition syntax for each system action, implementing an initial prototype system and using it to collect a data corpus. The data is then used to train the user and recognition models as described in Chapter 4. This section discusses the rest of the process, which consists of aspects specifically related to strategy learning.

5.2.1 Evaluation functions

An evaluation function has to be chosen to reflect the particular design goals for the dialogue system. The choice of this function will usually depend on the nature of the application. For example, transaction success is critically important in applications such as banking, while in some information dispensing systems it is possible to tolerate occasional failures in return for improving the dialogue flow.

The problem of selecting a suitable evaluation function was discussed in Section 2.5.2 and remains a matter of debate. However, the fact that the designer has the freedom to choose an evaluation function, or compare the results obtained using several different functions, is an advantage in that it allows the learning of strategies that are suited to the particular application. Also, after learning a strategy, it can be fine-tuned by adjusting the evaluation function and re-learning until the learned behaviour is appropriate. Subsequent user trials can then be used to elicit user satisfaction ratings which can be used to refine the evaluation function further.

The evaluation function should be easily measurable in terms of objectively available quantities. It should also correlate as closely as possible to what the designer really wants to optimise, which is usually taken to be user satisfaction. The experiments reported in this chapter use average transaction failure rate and average transaction duration, but other quantities such recognition performance could also be added.

5.2.2 State space representation

Next, a state representation is chosen for the application. This is a difficult and critical part of automatic strategy learning, and it is an advantage of the simulation-based approach that the state representation is only chosen at this late stage so that it can be changed relatively easily if the initially selected representation proves to be problematic. The state representation must be chosen such that knowledge of the current state provides sufficient information not only for selecting an optimal action, but also to enable the learning algorithm to estimate the expected future reward from that state. In addition, it should contain enough history information for the Markov assumption to be reasonable. At the same time, the size of the state space, which increases exponentially with the number of state variables, should be kept small enough for the problem to remain tractable.

Note that it is not necessary to represent all information about the dialogue state in this representation, only those aspects relevant to choosing the next action. For instance, in a slot-filling scenario where different slots are completely independent and the order in which they are to be requested is fixed, it is not relevant what the state of each slot is: only that of the current slot needs to be represented. Thus the state space in which the reinforcement learning agent is operating can be chosen to be a much simplified version of the full system state.

Once the state representation is fixed, the allowable actions for each state are specified. In principle, all actions could be allowed in any state, but this results in a space of dialogue policies that is unnecessarily large, causing training to be slow or intractable. Nonsensical actions can also result in unrealistic behaviour of the user simulation since it was only designed to model situations that occur in sensible task-oriented dialogue. Thus it is important to prune out those actions that do not make sense or are clearly suboptimal in a specific state. Again, the action set available to the learning agent may be more restricted than the full action set, with the conversion to detailed actions (intention level representations of utterances) being done by other components of the dialogue manager.

5.2.3 Learning the policy

Finally, the system is trained using the reinforcement learning algorithms presented in Section 3.4. The resulting policy can now be used as part of the dialogue manager, which will typically be responsible for wider aspects of the dialogue strategy as well. For instance, in a particular situation the policy might prescribe that the system should output a mixed initiative request for the next dialogue slot. The rest of the dialogue manager is then responsible for translating this to the appropriate set of intentions to be supplied to the utterance generator, and for supplying the appropriate syntaxes to the recogniser.

5.3 Artificial form-filling dialogue

The viability of the proposed approach was first investigated by means of an artificial form-filling application. It was constructed to have a conveniently small state space while being complex enough to allow a large number of different policies, many of which are more complex than those that human designers would typically implement.

5.3.1 Description of artificial application

The artificial application is an abstract representation of a form-filling dialogue, with a number of attributes for which values can be communicated to the system. Each attribute is represented by an arbitrary attribute name ($F1$, $F2$, etc.), with the allowable attribute values (represented by the symbols A, B, etc.) being specified along with any restrictions on when they are available.

In general, the range of values allowed for a particular slot can be a function of the current dialogue state, particularly the values of other attributes: a value of B for $F2$ might be disallowed if $F1$ is known to have value A. For instance, $F1$ might be a town name and $F2$ a cinema name. Then B might be "Warner Brothers Cinema", which may or may not be allowable depending on whether there is such a cinema in town A. Alternatively, in a system supporting different types of transaction $F1$ might be the transaction type and $F2$ a travel destination. Then the possible values for $F2$ will only be available if $F1$ is a travel-related transaction.

The experiments presented here use an example application with 3 attributes, $F1$, $F2$, and $F3$, of which the user is trying to communicate 2. The allowable values for each attribute are shown in Table 5.1. The $F1$ slot indicates the type of transaction, i.e. whether the user wants to specify $F2$ or $F3$. $F2$ and $F3$

Attribute	Values	Restrictions
F1	A, B	None
F2	goodC, badC	Both values available only if F1 has value A.
F3	goodD, badD	Both values available only if F1 has value B.

Table 5.1 *Allowable values for the different slots in the example artificial application.*

are restricted such that they can only be instantiated while $F1$ has a value of A or B respectively. (If attribute $F2$ or $F3$ is instantiated at a time when this does not hold, $F1$ is automatically assigned the appropriate value A or B.) The values assumed by $F2$ belong to a class of variables denoted by C, which in an actual application might correspond to, for example, a cash amount. However, the exact value within this class is not relevant from the perspective of the dialogue strategy (the system behaviour is desired to be independent of the exact amount). Thus the only information required to be represented here is whether the recognised value corresponds to that specified by the user: the values $goodC$ and $badC$ are used to denote, respectively, correct and misrecognised values of type C. Similarly, $F3$ assumes values of type D, with the exact content again being irrelevant except as regards whether it is correct or not.

Goal A:

Goal field	Value
F1:	A
F2:	C
F3:	-

Goal B:

Goal field	Value
F1:	B
F2:	-
F3:	D

Table 5.2 *User goals defined for the example artificial application.*

These values describe the dialogue state from the point of view of an omniscient observer: neither the system nor the user has direct access to it. In particular, they do not distinguish between values such as $goodC$ and $badC$, but group them together as C. It is therefore necessary to keep separate track both of the internal system and user states, and of the overall dialogue state. The application as described above allows two different user goals, illustrated in Table 5.2. Since the value of $F1$ (A or B) is what distinguishes these goals, they will be referred to as *goal A* and *goal B*. Possible interpretations for these goals in real applications are suggested in Section 5.3.4.

The aim of setting the problem up as described is to illustrate that the class of form-filling dialogues is broader than standardly implemented in artificial applications, where the different slots are assumed to be completely independent. The representation used here allows for dialogues that, as is often the case in real applications, display properties such as interaction between the different attributes. In the example such interaction is limited to some deterministic constraints on the values for certain attributes introduced by values of other attributes, which will not directly affect the action set. However, even such limited interaction can affect the eventual policy: the best way to obtain a value for $F2$ or $F3$ may well depend on whether or not $F1$ is instantiated. Also, the available action set may depend on such dialogue state dependent factors: for example, a system utterance such as “when do you want to travel to town A?” (indirect confirmation) is only available when the destination town is actually instantiated. It is important that such interaction be kept in mind while designing the state representation in Section 5.3.5.

5.3.2 Action set

Before designing the system strategy, it is necessary to define the actions available to the system. An action consists of system output followed by optionally waiting for input. The output is usually in the form of a system prompt directed at the user, e.g. “How may I help you?”, although other types of output such as database queries can also be defined. When obtaining input from a user, the system typically uses a recognition grammar which restricts the number of utterances that can be recognised. An action is thus defined by specifying both the system output and the possible inputs, specified by means of intention-level recognition grammars, that may follow. The effect of the input on the system state must be defined as part of the system state specification. Since a user model is to be used instead of real users, the effect of system output on user state must also be specified.

The following system actions were defined for the application. Unless specified otherwise, the system input is restricted to direct answers to the questions and function words such as “cancel” or “help”. Other answers will either be misunderstood or regarded as contentless.

1. Request $F1$ only.
2. Request $F2/F3$ only. No prior knowledge of $F1$ is implied: the value of $F1$ will be inferred if a value is specified for $F2$ or $F3$.
3. Mixed initiative: an open-ended request for $F1$, with the opportunity for the user to specify $F2/F3$ as well.
4. Implicit confirmation: state the currently understood value for $F1$ and request the corresponding $F2/F3$. The user can supply information consistent with the current information or give negative confirmation.
5. Confirm all fields explicitly. The user must answer yes or no.
6. Confirm $F1$ explicitly. The user must answer yes or no. Negative confirmation is interpreted as implying the alternative value of $F1$.
7. Terminate the dialogue: the system communicates the details of the completed transaction to allow evaluation of its success. No further input is allowed.

These actions were used to define two versions of the application. The first is a simplified version, where the explicit confirmation actions are disallowed. This restricts the system to the extent that only a few plausible strategies are available. These strategies can be compared independently of the learning algorithm so that the optimal strategy is known and can be compared to the learned policy.

The second (full) version is obtained by allowing the entire set of actions. It is to be expected that, when the system is allowed to use full confirmation, it will obtain a much lower average dialogue cost than with the simplified version and this is indeed what was observed. Allowing all the actions increases the number of plausible strategies so that it is much harder for a human to guess what the optimal strategy will turn out to be, and it is no longer practical to perform a separate simulation for each strategy. Thus in this case the learning algorithm is solving a problem to which the answer is neither known in advance, nor easily obtainable by other means.

5.3.3 Evaluation function

For these experiments, a simple evaluation function was chosen arbitrarily:

$$Cost = NumTurns + 100 \times DlgFailure \tag{5.1}$$

where *NumTurns* is the length of a dialogue in number of turns and *DlgFailure* is the number of failures (0 or 1) in the dialogue. The function was chosen so that the cost of a dialogue is dominated by a failure if it occurs, with the dialogue length being important only as a secondary measure to distinguish between failure-free dialogues.

5.3.4 Environment models

Since no data set is available for this artificial application, a number of hand-coded models were used for the user behaviour and recognition performance. This allows the differences in learned system behaviour to be investigated for different sets of model parameters. The different models correspond to scenarios one might encounter for different applications, user populations, etc., and were chosen such that the optimal dialogue strategy would be different for different scenarios, although it would not be possible for a human to predict this by inspection of the model statistics. The main purpose of these experiments is to demonstrate that the system learns different policies for different scenarios.

The scenarios implemented for these experiments are as follows:

- **Scenario A:** Baseline: C and D are not often confused with each other in situations where either could be specified. This might occur in an application where C and D are of completely different types, such as an amount of cash and a day of the week.

- **Scenario B:** The overall error rate is the same as for the baseline, but more of the errors are confusions between C and D. For example, C and D might be names of films and cinemas (a film name may be acoustically similar to another film name or sometimes to a cinema name).

- **Scenario C:** The $C \leftrightarrow$ *badC* and $D \leftrightarrow$ *badD* error rates are the same as for the baseline, but extra $C \leftrightarrow D$ confusion is present. This type of situation might occur if the day of the week in the baseline example is replaced by a numerical day of the month so that, while the probability of misrecognising the actual amount remains unchanged, there is an added chance of interpreting an amount as a day or vice versa.

- **Scenario D:** The confusion between C and D is even higher, but other error rates are lower than for the baseline. This might occur if the different possibilities for C are acoustically dissimilar, but correspond to similar possibilities for D, such as a source account and a destination account: the account names may be very different, but confusion can occur if the user does not specify clearly whether the account is specified as a source or destination.

For the case where confirmation is not allowed, only a small number of plausible strategies present themselves, as follows:

- **Strategy 1:** Long and safe. Request fields individually, without allowing mixed initiative.

- **Strategy 2:** Mixed initiative. Request fields individually, allowing the user to specify further fields.

- **Strategy 3:** Risky shortcut. Request $F2/F3$ immediately when $F1$ is unknown. $F1$ is not requested, but inferred from the value of $F2/F3$.

Scenario	**Strategy 1**			**Strategy 2**			**Strategy 3**		
	Length	**Failure**	**Cost**	**Length**	**Failure**	**Cost**	**Length**	**Failure**	**Cost**
A	3.62	0.20	23.6	2.79	0.29	31.8	**2.21**	**0.20**	**22.2**
B	**3.62**	**0.14**	**17.6**	2.81	0.22	24.8	2.21	0.25	27.2
C	**3.62**	**0.22**	**25.6**	2.78	0.32	34.8	2.21	0.34	36.2
D	3.45	0.05	8.5	**2.85**	**0.05**	**7.9**	2.22	0.25	27.2

Table 5.3 *Simulation results for selected strategies on the implemented scenarios.*

These strategies were implemented and a set of 1000 dialogues were simulated for each combination of scenario and strategy, to allow evaluation of the relative merits of the different strategies in different scenarios. The simulation results are presented in Table 5.3. The table shows, for each scenario-strategy combination, the individual cost components (average dialogue length in number of turns and average number of failures per dialogue) as well as the combined cost using the evaluation function of Equation 5.1. The highlighted statistics indicate the optimal strategy for each scenario.

In the experiments that follow, policies are learned for each of these scenarios, both for the case where confirmation actions are disallowed and the policy is expected to converge to the optimal strategies found above, and for the case where confirmation can be used and the optimal answers are not known in advance.

5.3.5 State space representation

With the application and action set having been defined, it is time to design a state space representation. This step defines the Markov decision process which is to be solved. Note that the state space is only chosen at a very late stage, so that it is always possible to change it, should the algorithm not perform as desired.

State variable	**Values**	**Meaning**
F1	Empty, A, B	Slot value.
F2	Empty, C	Whether the slot is filled.
F3	Empty, D	Whether the slot is filled.
InfoSource	None, 1, 2, 3, 4	Number of the action that resulted in the most recent piece of information.
Confirmed	No, Yes	Whether the contents of the currently instantiated attributes have been confirmed.

Table 5.4 *State space representation for the artificial dialogue application.*

The chosen state space representation is illustrated in table 5.4. Separate variables are used for the different slots, as well as single variables encoding the relevant dialogue history and confirmation status. This yields a total of 120 combinations of state variable values. However, only 25 of these are states that can actually occur in a dialogue (1 initial state, 12 states with $F1$ instantiated but $F2$ and $F3$ empty, and 12 states with $F2$ or $F3$ instantiated).

For the purpose of these experiments, the assumption is made that the state of every slot is relevant, so that all slots are included as state variables. In practice, this will rarely be the case: for many applications it will be possible to reduce the state space by identifying those slots of which the state is relevant when they are not the focus of the dialogue, and removing all others from the representation.

The variables describing individual slots encode only information that is relevant to the overall dialogue structure. Thus for $F2$ and $F3$ it is only relevant whether or not the slot has been filled with some value of type C or D; the exact value is not represented here.

The extra variables *InfoSource* and *Confirmed* are used to supply the history information required to ensure the Markov property. *InfoSource* is relevant since some actions lead to more reliable information than others. Thus information as to the most recent information-gathering action could impact directly on the optimal confirmation strategy, and is also required to distinguish between situations when the same short term result is obtained via different actions, to enable the algorithm to learn which of the actions is most rewarding in the long run. Another indication of the reliability of the information is the accompanying confidence measure. However, the handcrafted error models in these experiments did not include confidence measures, so they were not used in the state space.

The *Confirmed* variable is only used in the full version of the problem, when confirmation actions are available. Confirmation information is restricted to a single variable, since the confirmation actions that have been defined always confirm all information that has been collected in the dialogue. If confirmation actions for individual slots were available, the state space would be expanded to reflect this. Note that the state space representation is heavily dependent on application-dependent factors such as the available actions and dependence between different units of information.

It is also necessary to specify which actions are available for each state. Rather than allow all actions in every dialogue state, actions that would be nonsensical are removed, along with actions that are clearly suboptimal, such as asking for confirmation when all actions have already been confirmed or terminating the dialogue when a complete transaction has not been specified. This makes a large difference to the size of the state-action set and thus the tractability of the algorithm: the state-action set was reduced from 175 (allowing all actions in each of the 25 states) to 51, an average of only about 2 actions per state. The largest number of actions deemed reasonable in any one state was 3, e.g. actions 1, 2 or 3 for the initial state, or actions 2, 4, or 6 in states where $F1$ is instantiated but unconfirmed while $F2$ and $F3$ are uninstantiated. Also, states in which the instantiated slots have been confirmed and correspond to a valid goal are forced to use the termination action. Restricting the state-action set in this way resulted in a speedup of algorithm convergence by a factor in the order of 100.

5.3.6 Experiments and results

5.3.6.1 Learning parameters

Since the artificial application is simple enough for convergence to be unproblematic, only straightforward Q-learning was used in these experiments. The step size parameter α_k was controlled by an exponentially decreasing schedule:

$$\alpha_k = 1/(p^k), \tag{5.2}$$

where k is the number of times a particular parameter has been updated, and p is a constant between 0.5 and 1.0. Preliminary experiments gave good convergence with $p = 0.8$, which is the value used in these experiments. Exploration was done using an epsilon-greedy strategy, with epsilon starting at 1 (random exploration) and decreasing exponentially to a minimum of 0.1 (exploration occurs 10% of the time). These settings satisfy the theoretical requirements to ensure convergence to the optimal policy (Section 3.4.5).

5.3.6.2 Learned policies using the simplified action set

The first set of experiments was to learn policies for the simplified (no confirmation) problem, using the various scenarios. The learned policies are expected to be those found via simulation to be optimal (highlighted in Table 5.3). The Q-learning algorithm was run for all 4 scenarios, and in each case it converged to the predicted policy, or a policy equivalent to it in the sense that it only differs for states that are not in fact visited when the learned policy is followed. In all cases, convergence takes fewer than 50000 iterations.

5.3.6.3 Learned policies for the full application

Next, policies were learned for the full application, using the same scenarios. Again the policies converged with convergence times varying between 5000 and 50000 iterations. In all cases, the learned policy uses confirmation to improve on the average reward obtained when confirmation was not available, and seems reasonable when inspected. The use of mixed initiative and implicit confirmation vary from one scenario to another.

In order to evaluate the learned policies, a set of predesigned policies, such as might be chosen by a human designer, was implemented. The predesigned policies use a number of approaches to confirmation and mixed initiative, but use such approaches uniformly, rather than adapt them according to the particular dialogue slot under consideration. They can be summarised as follows (building on the strategies described in section 5.3.4):

- **Policy 1:** Long and safe strategy, with a single full confirmation at the end of the dialogue.
- **Policy 2:** Mixed initiative strategy, with a single full confirmation at the end of the dialogue.
- **Policy 3:** Risky shortcut strategy, with a single full confirmation at the end of the dialogue.
- **Policy 4:** Long and safe strategy, with step by step confirmation of details.
- **Policy 5:** Mixed initiative strategy, with step by step confirmation of details.
- **Policy 6:** Risky shortcut strategy, with confirmation of $F1$ only. (Although this policy performed poorly in these experiments, it may be sensible in cases where recognition performance for $F2$ is high, since more can be deduced from a negative response if only $F1$ is being confirmed.)

Since the scenarios under investigation are uniform with respect to the different slots, the learned policies can be expected to be similar to some of the human-designed strategies: in fact, the learned policies for scenarios 2, 3 and 4 are identical to policies 3, 1 and 2 respectively. The learned policy for scenario 1 is close to policy 3, but differs in that it prefers action 1 to action 3 in some contexts when only $F1$ is known: this difference does not lead to a significant difference in performance.

In further experiments it was found that altering the scenarios to use non-uniform (but still realistic) recognition statistics does lead to non-uniform policies being learned. However, these policies still did not yield average costs that differ significantly from the predesigned strategies, possibly due to the exaggerated simplicity of the application. A more complex application would be required to demonstrate interesting non-uniform policies.

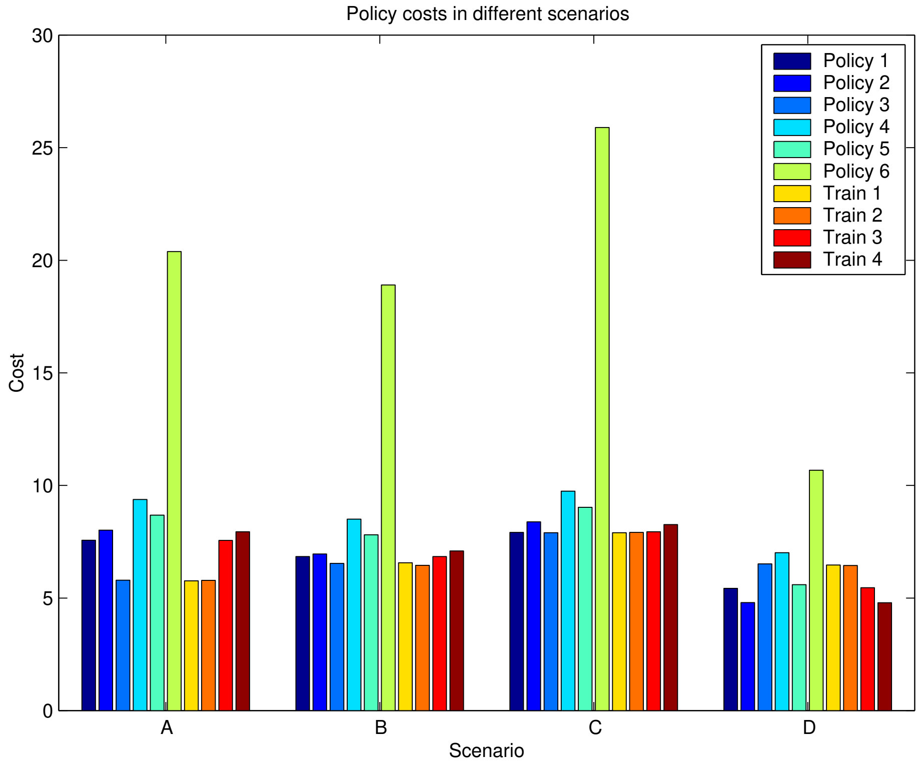


Figure 5.3 *Average costs of various dialogue strategies in different scenarios.*

The average costs of the learned and predesigned policies were measured by simulation over 100000 dialogues and are compared in Figure 5.3, where "Train 1" to "Train 4" refer to the policies obtained after training on scenarios 1 to 4. The first thing to notice is that, when evaluated for a particular scenario, the policy trained on that scenario performs at least as well as any policies trained on other scenarios, as well as the hand designed policies. This is what is expected if the policies being learned are optimal for the scenarios on which they are trained.

It can be seen that picking the best of the predesigned policies for each scenario gives equivalent performance to using the learned policies: this is because they are identical or nearly identical. However, the predesigned strategies that do well differ from scenario to scenario, and it is not possible to predict which predesigned strategy would do well.

For instance, one might imagine that Policy 3, supplementing the shortcut strategy with a confirmation step at the end is a sensible policy to follow, which turns out to be the case for three of the four scenarios under investigation. Yet it performs poorly in the fourth scenario. Similarly, Policy 6 performs very poorly on all four scenarios, yet it should be possible to construct a scenario where it is optimal.

5.3.6.4 Discussion

These experiments demonstrate that the system learns dialogue strategies that vary as the scenario (defined in terms of statistics of user actions and recognition performance) changes. In the simple cases where the optimal dialogue strategies are known, the learned strategies agree with the optimal ones. In more complex cases, where the optimal strategies are not known in advance, the learning algorithm provides a means of finding them. The learned strategies improve on randomly chosen ones, and seem reasonable on inspection.

The algorithm is able to learn policies that treat different slots differently, and take account of the dialogue history. It chooses approaches to mixed initiative and confirmation on the basis of statistics which are not easily processible by human designers.

Of course the results are generated using the same user model that was used for learning the strategies in the first place and can therefore not be regarded as a final test. However, they do give an indication that is valid to the extent that the user model is accurate and serve as a useful substitute until full field trials can be conducted.

5.4 Application to a real dialogue system

The experiments on the artificial application confirmed that it is possible to use the proposed approach to learn strategies that are optimal and depend on the specific situation. Next, it was tested on a more complex real world application: the cinema information system used in Chapter 4. Since this system was originally developed by a team of dialogue design engineers and refined via user trials, it provides a challenging baseline for evaluation of the learned strategy.

5.4.1 Description of application

The full cinema information system has more than one independent sub-dialogues that are completed in succession, and offers the user the possibility of completing several consecutive goals. However, the experiments described here are constrained to the central part of the dialogue, where the user is allowed to specify a goal which has one of four different types: film listings, film distribution, film times and cinema address. All of these goals can be described in terms of a simple slot-filling dialogue, where the first slot is the goal type and which other slots are to be filled depends on the specific transaction. The user and error models presented in Chapter 4 were used to simulate the application environment.

5.4.2 Action set

Action	Description
Mixed Initiative	Mixed initiative query for more information (available if current slot is empty).
System Initiative	Query for information on the current slot (available if current slot is empty).
Explicit confirmation	Confirm the contents of the current slot explicitly.
Confirm all	Confirm the contents of all slots.
Implicit confirmation	Confirm the contents of the current slot implicitly while querying for information on the next slot.
Accept	Accept information in the current slot without confirmation at present. Terminate if a complete transaction has been specified.

Table 5.5 *Actions defined for the cinema application.*

Some decisions that need to be taken in the course of the dialogue, such as deciding which slot should be in focus at a given moment, or which syntax to use after deciding to do a system initiative query on the current slot, are easy to design and can be implemented by hand. It is therefore only necessary to define a restricted set of actions to make available to the learning system. These actions determine the extent of the strategy learning problem, and the amount of freedom available in choosing between different policies. The action set chosen for the cinema application is summarised in Table 5.5. It allows for automatic design of the choice between mixed and system initiative and the confirmation strategy (a choice between explicit confirmation, implicit confirmation, and delaying confirmation until later).

5.4.3 Evaluation functions

As discussed in Section 5.2.1, the designer is free to choose an arbitrary evaluation function to be optimised. These experiments were performed for four evaluation functions of the form

$$Cost = NumTurns + FailCost \times NumFailures \tag{5.3}$$

where *NumTurns* is the average number of turns per transaction, *NumFailures* is the average number of failures per transaction, and *FailCost* is a weighting factor to which the values 20, 10, 5 and 2 are assigned.

These evaluation functions represent a range of scenarios where the system is expected to learn dialogue strategies ranging from safe (when the cost of a failure is high, it pays to use confirmation thoroughly) to risky (when failures are less crucial, it is better to streamline the dialogue and use confirmation only when really necessary). A comparison of the policies learned for the different criteria will illustrate how the learning algorithm adapts to these different scenarios in a way that is not easy for human designers.

The choice of values for $FailCost$ is motivated by simulation experiments performed with the manually designed baseline system. It was found that this system performed well in the middle of the chosen range, but poorly for values outside it. Thus, using the $FailCost$ value in Equation 5.1 would result in a comparison that is unfair to the baseline. On the other hand, this illustrates the inflexibility of manually designed systems: if it were decided that the failure rate is too high, an automatically designed system could simply be retrained using a larger value for $FailCost$, but changing a manually designed strategy to reduce the failure rate is much more difficult.

5.4.4 State space representations

Variable	**Possible values**	**Description**
CurrentFocus	Type, Day, Film, Cinema	Which slot of the dialogue is currently in focus.
Status	Empty, Filled	Whether the current slot is empty or filled (unconfirmed).
#Empty	0, 1, 2, 3	How many of the not-in-focus slots are currently empty.
#Unconfirmed	0, 1, 2, 3	How many of the not-in-focus slots are currently unconfirmed.
InfoSource	Default, Negated, Elicited, Unelicited	How the information in the current slot was obtained.
Confidence	Default, 0.2, 0.3, $\cdots$, 1.0	Confidence score of the information in the current slot.
ConfLevel	Default, Low, High	Thresholded confidence of the information in the current slot.

Table 5.6 *Variables used in the different state space representations.*

Since the requirements for the state space representation (see Section 5.2.2) are not easy to satisfy, it is useful to be able to experiment with different state space representations. The variables summarised in Table 5.6 were used to construct a number of different state spaces. These are listed in Table 5.7, along with the corresponding state-action space sizes. The latter are equal to the number of Q-values that need to be stored after impossible states and state-action combinations have been pruned out.

All of the representations rely on the idea of restricting the state space size by retaining detailed information only for the slot that is currently in focus. Infor-

Name	**State variables**	**Size of State-Action space**
Small	CurrentFocus, Status, #Empty, #Unconfirmed	209
InfoSource	CurrentFocus, Status, #Empty, #Unconfirmed, InfoSource	480
LargeConf	CurrentFocus, Status, #Empty, #Unconfirmed, Confidence	1298
SmallConf	CurrentFocus, Status, #Empty, #Unconfirmed, ConfLevel	330
SmallConfInf	CurrentFocus, Status, #Empty, #Unconfirmed, ConfLevel, InfoSource	778

Table 5.7 *State space representations.*

mation about other slots is hidden from the reinforcement learning algorithm, while the algorithm for deciding which slot should be in focus at any given moment (based on this hidden information) is simple enough to be handcrafted.

Note that the variables defined in Table 5.6 are application independent: they can be used for any form-filling dialogue. This is important in facilitating cross-domain portability of the approach.

The simplest state space representation (*Small*) does not store any knowledge related to the reliability of the information in the current slot. Such knowledge is added in the other representations, at the cost of enlarging the size of the state space. The *ConfLevel* variable used in *SmallConf* and *SmallConfInf* was designed after studying the policies learned for the *LargeConf* state space, where it was observed that these policies tended to base their actions on a single confidence threshold. In this way it is possible to keep the state space size down without degrading performance.

5.4.5 Experiments and results

5.4.5.1 Baseline systems

The main baseline system available is the original manually designed application. This uses a large number of internal system variables and advanced strategies such as variable confidence thresholds that depend on the number of times specific fields have been prompted for, etc. Since, in order to keep the design problem tractable, the automatically designed policies were restricted to simple state spaces, comparing with this full system is not entirely realistic. For this reason several handcrafted systems that operate in the *Small* state space were constructed using the action set of Table 5.5. These policies are simple, not taking much dialogue context into account, but of the type that a human would implement. They are summarised in Table 5.8.

Policy	Description
Handcraft 1	Mixed initiative with full confirmation at each step.
Handcraft 2	Mixed initiative with explicit confirmation of each slot but no full confirmation at the end.
Handcraft 3	Mixed initiative with explicit confirmation of each slot and full confirmation at the end.
Handcraft 4	Mixed initiative with confirmation only at the end.
Handcraft 5	System initiative with full confirmation at each step.
Handcraft 6	System initiative with explicit confirmation of each slot but no full confirmation at the end.
Handcraft 7	System initiative with explicit confirmation of each slot and full confirmation at the end.
Handcraft 8	System initiative with confirmation only at the end.

Table 5.8 *Handcrafted policies.*

5.4.5.2 Training parameters

Training was performed using Watkins' $Q(\lambda)$ algorithm, as presented in Section 3.4. The step size and exploration parameters were set as in Section 5.3.6.1.

An effect of using eligibility traces was that policies did not stabilize early: since a number of Q-values usually change simultaneously in each update, the policy is less stable and changes after almost every iteration, even after training for as many as 10^7 episodes. However, when sufficient training has been performed for the Q-values to be within the desired margin of accuracy, the policies between which the algorithm is switching are similar enough to yield results which are correspondingly close.

It is interesting that this effect does not occur when simple Q-learning is used: in this case the policy remains unchanged for increasingly long periods, with a stable period of 10^5 episodes typically being observed before the 10^6th training episode. This can be misleading, since convergence has in fact not taken place at such a point. A better criterion for determining a proper training length is therefore the convergence of the Q-values. Figure 5.4 shows how the Q-value of the initial dialogue state progresses in a typical training run. After 10^6 episodes the fluctuations are in the order of 0.01, which is sufficient for the purpose of these experiments. The training runs for results presented in the remainder of this chapter were performed for a length of 10^6 episodes. To verify the accuracy of these experiments, a number of training runs were also performed for periods of up to 10^7 episodes, with no noticeable change in results.

Policies trained using various values of the eligibility trace decay parameter λ are compared in Figure 5.5. This illustrates the effect of changing the learning algorithm from Q-learning ($\lambda = 0$) to Monte Carlo learning ($\lambda = 1$) via a set of intermediate algorithms. The graph shows the performance of policies using each of the state space representations, for the case of $FailCost = 20$. Other evaluation functions gave similar results.

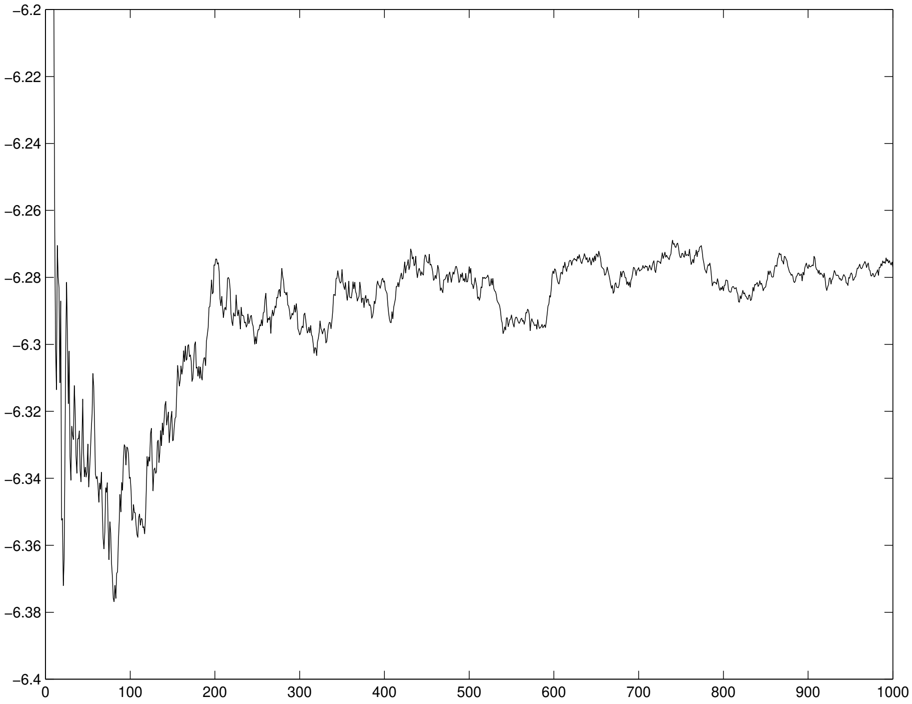


Figure 5.4 *Q-value of the initial dialogue state as training progresses.*

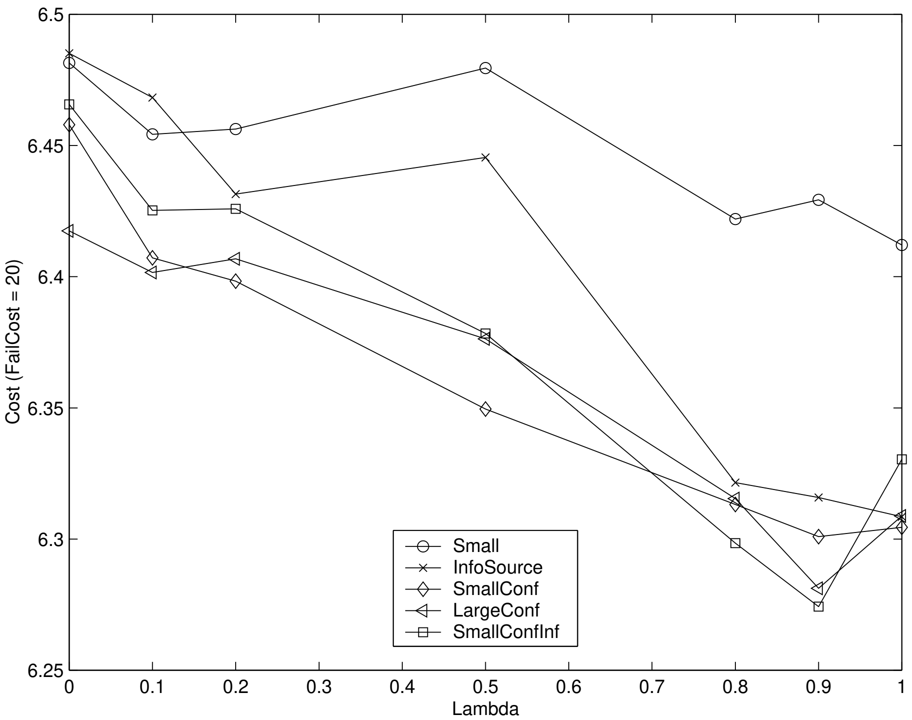


Figure 5.5 *Comparison of policies trained using different eligibility trace decay rates.*

It can be seen that different values of λ work well for different state space representations, but overall the best results are obtained for algorithms close to Monte Carlo learning rather than Q-learning, with $\lambda \approx 0.9$. This supports the observation reported in [112] that a value of $\lambda = 0.9$ worked well over a range of problems where memoryless policies were obtained for partially observable MDPs. (Note, however, that the difference in performance for values close to 1 was small and probably not significant. Thus it is not clear whether the use of eligibility traces gives any real improvement over simple Monte Carlo learning.) The remaining experiments presented here use this value for λ.

5.4.5.3 Learning experiments

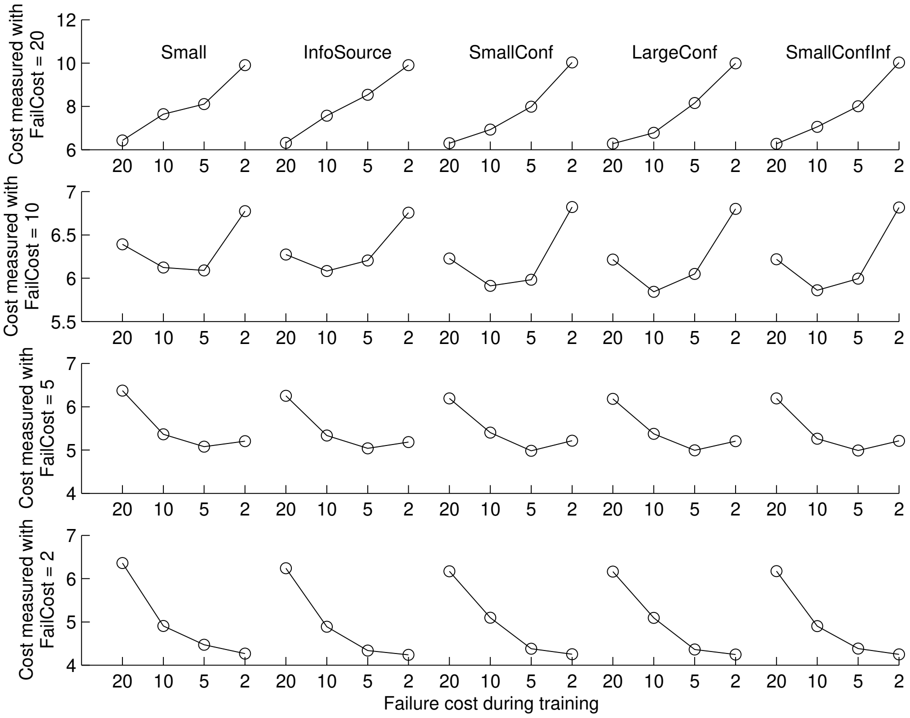


Figure 5.6 *Comparison of policies trained on different evaluation functions for all state space representations: each graph plots the cost for all the policies, measured using the evaluation function indicated on the left.*

After training policies for each of the different state space representations and evaluation functions, the performance of each policy was measured by means of simulations (10^5 episodes per evaluation). It should be pointed out that this method of evaluation is not ideal, since performing full field trials is the only definitive method of evaluating a dialogue system. However, since field trials are expensive, time consuming and often impractical, evaluation by simulation is a useful substitute.

The results are shown in Figure 5.6. Each graph represents the cost, shown on the vertical axis, as measured using a different evaluation function which is indicated by the label on the left. The horizontal axes represent the policies learned using the different algorithms, which are grouped according to the state space representation used (indicated at the top). The solid lines join policies learned using the same state space representation but different values of *FailCost*.

For each evaluation function, the policies optimised for that function can be expected to outperform those optimised for a different function. Thus, on the first graph ($FailCost = 20$), the policies trained using a *FailCost* of 20 should outperform those using other values. The graph shows that this is indeed the case, and also that the difference in performance between policies optimised for different evaluation functions is much greater than that between policies which use different state space representations.

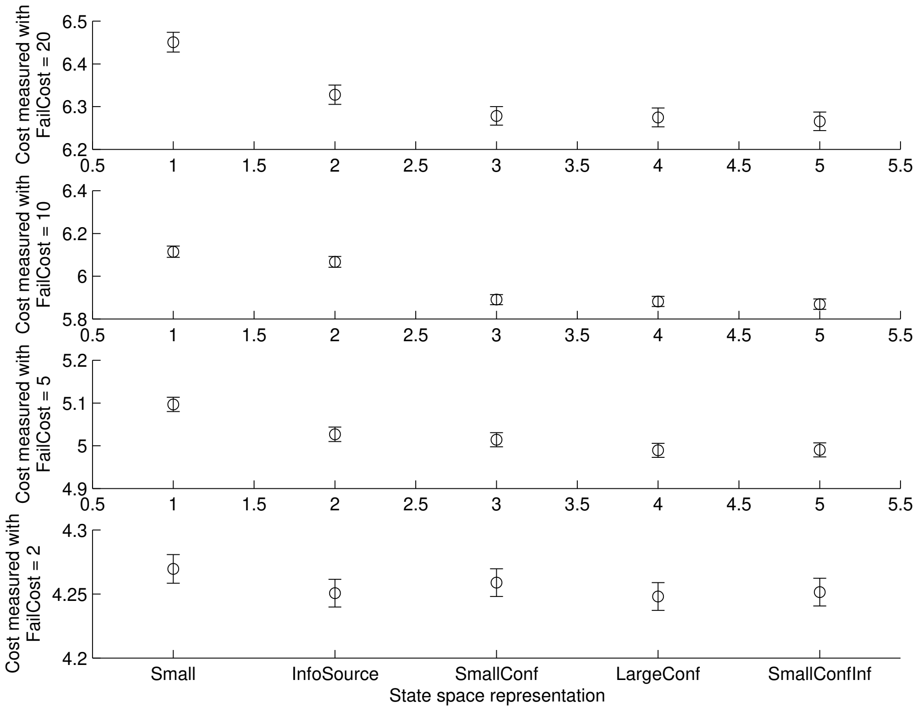


Figure 5.7 *Comparison of optimised policies for all state space representations: each graph plots the cost for policies trained and evaluated using the evaluation function indicated on the left; error bars indicate* 95% *confidence intervals.*

Figure 5.7 provides a closer look at the differences between different state space representations, showing only the policies that were optimised for the evaluation functions they are being evaluated on. The evaluation precision is indicated by error bars representing 95% confidence intervals. It is seen that, although the difference in performance between different state space representations is small, the larger state spaces (where the dialogue manager has more state information available) tend to outperform the smaller ones. Also note that some evaluation functions make the design task easier than others: for the case of $FailCost = 2$ a risky strategy with very little confirmation is called for, so that information on the reliability of user-provided information becomes less relevant (any confidence above the minimum recognition threshold will usually be sufficient). This means that, for this evaluation function, the smaller state spaces are not harmed as much by the lack of information and can do as well as the larger ones.

Policy	**Average cost**			
	FailCost = 20	**FailCost = 10**	**FailCost = 5**	**FailCost = 2**
Small	6.43	6.12	5.08	4.27
InfoSource	6.32	6.08	5.04	4.24
LargeConf	6.28	5.84	5.00	4.25
SmallConf	6.30	5.91	4.98	4.25
SmallConfInf	6.27	5.86	4.99	4.25
Handcraft 1	9.36	8.79	8.50	8.33
Handcraft 2	11.42	9.99	9.27	8.84
Handcraft 3	9.41	8.78	8.47	8.28
Handcraft 4	9.51	8.84	8.50	8.30
Handcraft 5	8.33	7.75	7.46	7.28
Handcraft 6	10.76	9.27	8.52	8.08
Handcraft 7	8.74	8.04	7.67	7.48
Handcraft 8	8.61	7.77	7.34	7.09
Full hand design	6.82	5.52	4.88	4.49

Table 5.9 *Costs obtained for automatically designed and handcrafted systems.*

Table 5.9 compares the results for the optimised policies obtained using the different state space representations, as well as the handcrafted policies and the original hand designed system. Due to the nature of the experiment, where the number of dialogues on which the evaluation is performed is chosen to be large, even small differences between the numbers in this table are statistically significant. This was investigated by means of t-tests, which showed differences of more than 0.04 to be significant at a level below $p = 0.05$, while smaller differences were not significant at this level.

The automatically designed systems are relatively close to each other in performance, all of them outperforming the handcrafted policies by large margins.

The handcrafted policies of particular interest are Handcraft 5 and Handcraft 8, which outperform the others. Handcraft 5 follows the commonly recommended safe strategy of using system initiative while confirming everything fully at each step. It can be compared with Handcraft 1, which uses the same confirmation strategy but replaces system initiative with mixed initiative. For this application, the use of mixed initiative actions results in a deterioration of performance on all evaluation functions.

Handcraft 8 uses system initiative but follows a more risky confirmation strategy than Handcraft 5, resulting in better performance when $FailCost$ is low, but worse performance when $FailCost$ is high. However, neither of these policies come close to the performance of the automatically learned policies, due to the fact that they do not vary the treatment of different slots according to their indi-

System:	What information would you like?
User:	Can you give me information about the Odeon?
System:	[Accepting the default assumptions that the transaction type is film times and the day of interest is today.] Which film do you want to see?
User:	Angel Eyes. [Misrecognised, but not confirmed.]
System:	A.I. is on at...

Figure 5.8 *Typical dialogue with a risky learned strategy. Intentions have been expanded to the word level in order to construct full sentences. Comments are added in square brackets.*

vidual statistics.

The learned policies outperform the original design on some evaluation functions but are beaten on others, highlighting the inflexibility of the hand design which cannot be optimal for all evaluation functions. The fact that the manually designed system performs as well as the automatically learned systems is not surprising, since it has access to many variables not represented in the state spaces used by the reinforcement learner and effectively acts in a much larger state space. It is encouraging that, despite this, the learned policies sometimes outperform the full system. It is to be expected that the performance could be improved further by incorporating the learned policies into the manual design.

5.4.5.4 Comparison of learned policies

The different evaluation functions represent a range of scenarios where the system is expected to learn dialogue strategies ranging from safe (when the cost of a failure is high, it pays to use confirmation thoroughly) to risky (when failures are less crucial, it is better to streamline the dialogue and use confirmation only when really necessary). Figures 5.8 and 5.9 show some examples of typical dialogues produced with policies learned using failure costs of 2 and 20 respectively. It can be seen that the strategy in Figure 5.8 produces dialogues that are short but prone to error, while the strategy in Figure 5.9 produces longer dialogues with more robustness against errors.

The histogram in Figure 5.10 shows how the relative frequencies of the different actions in the *Small* state space change as the cost of a transaction failure decreases from 20 to 2. While the exact behaviour of a policy cannot easily be predicted, the large scale trends conformed to what one would expect: as the cost of failure decreases, confirmation actions are used less frequently while acceptance actions increase. This illustrates how the learning algorithm adapts to different scenarios in a way that is hard for human designers to emulate.

Figure 5.11 illustrates how these different policies produce different dialogue costs. The histograms show the cost distributions obtained when the four policies of Figure 5.10 are evaluated using $FailCost = 5$. The first two policies were optimised for $FailCost = 20$ and $FailCost = 10$ respectively. They have high average dialogue length, which produces a noticeable shift to the right, as well

Dialogue 1:	
System:	What information would you like?
User:	What is on at the Odeon?
System:	Do you want to know which films are showing at the Odeon today? [Full confirmation.]
User:	Yes.
System:	The films are ...
Dialogue 2:	
System:	What information would you like?
User:	What is on at the Odeon? [Cinema name misrecognised.]
System:	Do you want to know which films are showing at Warner Brothers today?
User:	No.
System:	What information would you like?
User:	What is on at the Odeon?
System:	Which day do you want information for?
User:	Today.
System:	Do you want to know which films are showing at the Odeon today?
User:	Yes.
System:	The films are ...

Figure 5.9 *Typical dialogues with a safe learned strategy. Intentions have been expanded to the word level in order to construct full sentences. Comments are added in square brackets.*

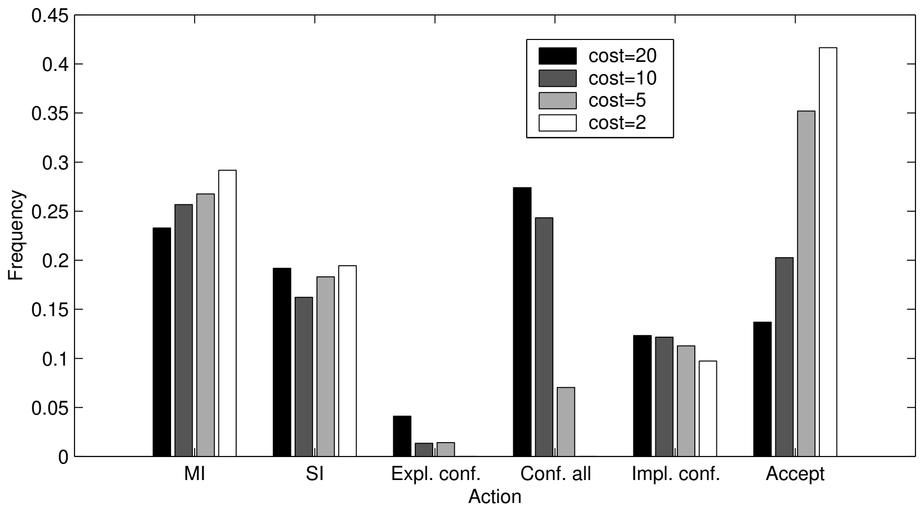


Figure 5.10 *Relative frequencies of actions in policies optimised for different evaluation functions.*

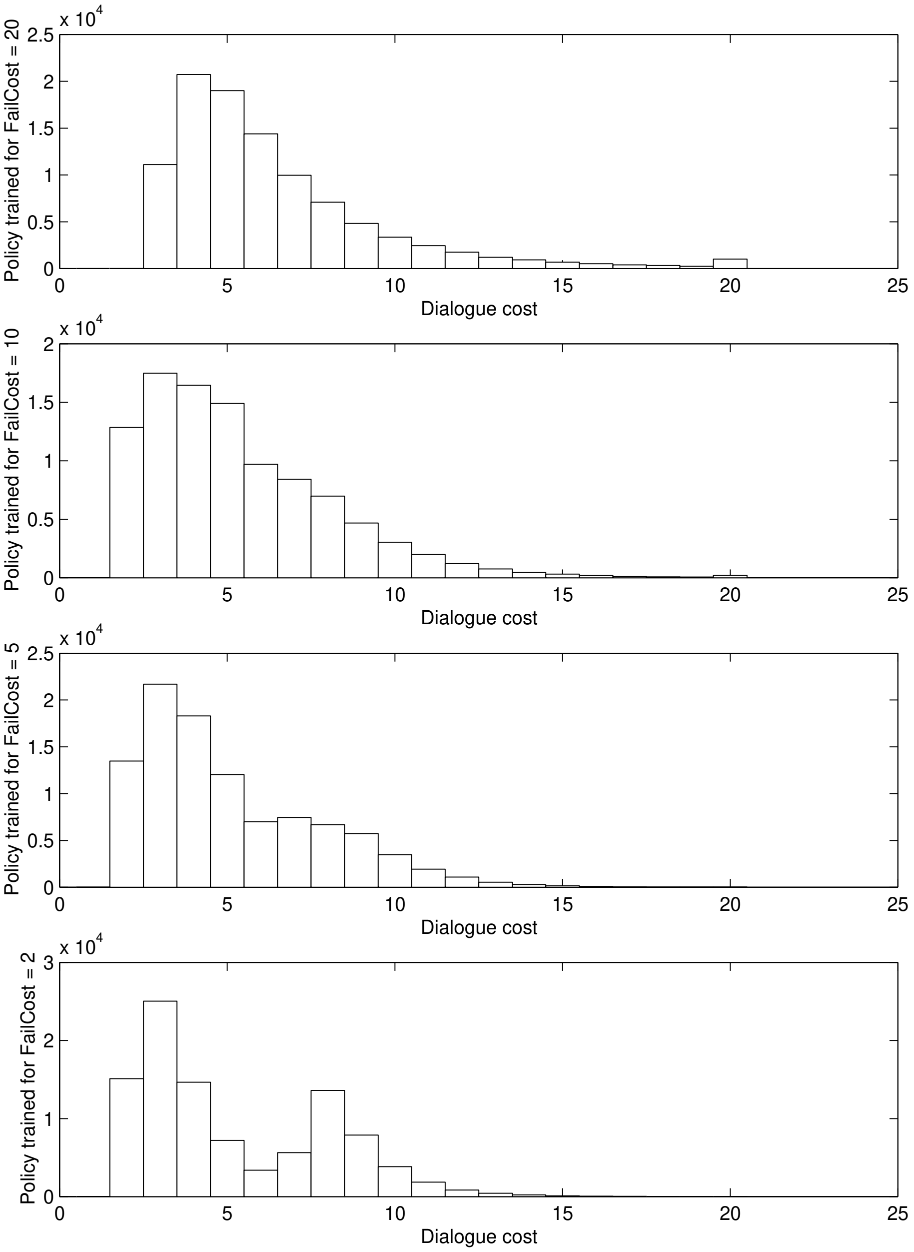


Figure 5.11 *Distribution of dialogue costs for policies optimised for different evaluation functions: each graph shows a histogram of costs (evaluated using $FailCost = 5$) for policies trained using the evaluation function indicated on the left.*

as long tails reflecting the existence of exceptionally long dialogues. These correspond to dialogues with multiple confirmation sub-dialogues containing recognition errors.

The third policy is the one that has been optimised for this particular evaluation function. It strikes a balance between minimising the average dialogue length and restricting the transaction failure rate, which is visible as a secondary peak emerging to the right. The fourth policy was optimised for $FailCost = 2$ and shortens dialogue length even more, moving the main peak further to the left. However, this comes at the cost of increasing the failure rate such that the secondary peak causes the overall dialogue cost to rise.

Policy	**Component 1**			**Component 2**		
	Mean	**Variance**	**Weight**	**Mean**	**Variance**	**Weight**
FailCost = 20	5.00	1.97	0.71	9.80	21.73	0.29
FailCost = 10	3.69	1.55	0.56	7.51	8.84	0.44
FailCost = 5	3.48	1.17	0.62	7.73	5.50	0.38
FailCost = 2	3.11	0.77	0.57	8.13	3.89	0.43

Table 5.10 *Parameters of two component mixture model fitting the data in Figure 5.11.*

The bimodal nature of the data in Figure 5.11 suggests fitting a two component mixture model to it, to gain further understanding of the separate contributions of successful and failed dialogues to the average dialogue costs. This analysis was performed by estimating a two component mixture Gaussian model using the EM algorithm. The means, variances and weights of the two components are shown in Table 5.10. The behaviour of these parameters confirms the observations made by inspection of Figure 5.11: as the policies become riskier, the mean of the main peak becomes smaller, reflecting the shorter average dialogue length. At the same time, the secondary peak changes from a high variance component modelling the long tail of the distribution to a low variance component modelling the more pronounced secondary peak, reflecting the larger number of failed dialogues.

5.5 Summary

This chapter described how the user and error simulation tool presented in Chapter 4 can be used instead of a direct state transition model to design dialogue strategies in an MDP reinforcement learning framework.

One of the big problems in automatic strategy learning is designing a state space representation that includes all the necessary information, allows the Markov assumptions to be made, and is nevertheless small enough for the problem to remain tractable. Since the state space representation can be difficult to design, a great advantage of the simulation approach compared to previous work using state transition models is the ability to explore different state spaces using a single fixed dialogue corpus, transcribed without reference to system state. Since state space design has to be performed every time a new application is developed, it is important that it is made as convenient as possible.

Preliminary experiments on an artificial application demonstrated that the approach can learn optimal policies in a large policy space and that the learned behaviour depends on the statistics of the specific application, as estimated by the user and error models.

The approach was applied to learn dialogue strategies in a real application, automatically selecting among the many possible approaches to confirmation and mixed initiative issues. A number of state space representations were compared, as well as the effect of using different cost functions. On evaluation in simulation experiments, the learned policies outperformed handcrafted policies using the same state space, and gave performance similar to that of the original hand design, which used a much richer state space and action set. They also display flexibility, optimising their behaviour according to the cost function being used.

While the automatic design outperforms manual designs in a fixed, small state space, the original hand design has the advantage of being able to use a larger number of state variables. While some of these may be useful to incorporate into the reinforcement learning state space, the learning problem would become intractable if all of them were used. It should, however, be possible to combine the respective strengths of the manual and automatic designs by incorporating the automatically learned strategy into the full design, thus obtaining further improvement.

6

Conclusions

This thesis presented two novel contributions: a quantitative approach to dialogue simulation that can be used as a tool for evaluation and development of dialogue systems, and the application of this tool in a system for automatic dialogue strategy learning. This chapter will discuss some implications of the research that has been presented and make some suggestions for future work.

6.1 Dialogue simulation

The dialogue simulation tool developed in Chapter 4 allows quantitative simulation of dialogues based on statistics extracted from a corpus of dialogues between users and a prototype system. The simulation differs from existing approaches in that it combines probabilistic, data-driven methods with a simple structural model of user state. Previous simulation systems used either purely probabilistic methods, which can lead to inconsistent dialogues, or purely deterministic methods, which do not attempt to make quantitatively accurate predictions of user behaviour.

The accuracy of the simulations was evaluated by comparison with real dialogues, which corresponded closely to the simulated dialogues in terms of average transaction times. While this does not imply that the simulated dialogues resemble real dialogues in every respect, it does indicate that reducing the duration of simulated dialogues is likely to result in corresponding reductions in the duration of real dialogues.

The fact that the model is sensitive to variations in the input data encourages the use of different models for different user populations (such as experienced users and complete novices) where separate data is available. This would allow a dialogue system to be optimised for different types of users, or user populations consisting of combinations of the different user types. It also opens the possibility for designing adaptive systems that estimate user expertise online and adapt their strategy accordingly.

6.2 Strategy learning

Researchers have long been debating the question of which dialogue management strategies are the "best". The recent trend has been to acknowledge that there exists no definitive answer to this question: the optimal dialogue strategy depends on the specific application scenario. Instead, the emphasis of the debate has shifted to methods for adapting dialogue strategies to the specific situation. To quote a typical opinion in this regard: "...we conclude the necessity to design a robust and flexible system suitable to have different dialogue control strategies depending on the characteristics of the user and the performance of the speech recognition module" [87].

It is this problem that the strategy learning approach addresses, by providing a principled means of finding solutions for difficult design problems such as how to manage confirmation and initiative. The learned strategies are adapted to the user behaviour and recognition performance as measured in the particular application, as well as the evaluation function defined for the application.

The resulting strategies were found to outperform standard handcrafted approaches in simulation experiments. This is mainly due to their ability to treat different parts of the dialogue, such as the various slots in a form-filling dialogue, differently. Their simulated performance also outperformed that of the full manual design on some performance measures, in spite of having to act in a much more restricted state space. While it is not envisaged that policies acting in a restricted state space will replace unrestricted manual designs, this opens many possibilities for improving such systems.

The approach taken here, which makes use of a dialogue simulation instead of a state transition model, enables the designer to experiment with different state spaces easily. This is a big advantage, since the use of automatic learning causes state space design to replace manual strategy design as a major challenge in the development process. Making the design of a good state space easier is a big step towards cross-domain portability.

In a real application it will often not be clear what the best choice of cost function is. Ideally, a cost function should be constructed based on user satisfaction studies done in the specific application domain, using methods such as the PARADISE framework [63]. Where such studies have not been done, the designer may have to choose a cost function in an ad hoc way. This is not as disadvantageous as it may seem, since the behaviour of the learned strategy can then be evaluated qualitatively and the cost function adjusted until the learned strategy displays the desired characteristics. Subsequent user trials can then be used to elicit user satisfaction ratings which can be used to refine the cost function further.

It is sometimes stated that online learning of strategies is an important area of future research. However, while there may be some scope for developing methods that use reinforcement learning in the course of a conversation, the presently available methods are quite adequate for coping with potential non-stationarity in the user population. By logging the dialogues conducted with users while the system is in operation, fresh data can be obtained with which to retrain the models on a regular basis. Whenever the models have been retrained, the reinforcement learning algorithm can be run to obtain strategies that have been optimised according to the most recent trends in user behaviour.

The main weakness of the experiments presented here is that they do not include a full field trial evaluating the results on real users. While this would of course be desirable, the results generated using simulations are nevertheless informative. The quality of the results depend on the quality of the simulations, which have been evaluated by comparing them with real dialogue data. Also, previous experiments that used field trials to evaluate the results of reinforcement learning approaches [101, 102] demonstrated that these methods can give good results when tested by real users.

6.3 Suggestions for future work

6.3.1 Evaluation of results by means of field trials

An important experiment that remains to be done is the evaluation of the results produced here by means of field trials involving real users. This is the only way to produce a final verdict on the accuracy of the simulations used for evaluation in this thesis. Unfortunately, such experiments are expensive and time-consuming, so that it could not be performed in the course of the research described here.

6.3.2 Integration of automatic design with other design techniques

Since the automatically learned strategies yield performance similar to the manually designed system, it would be possible to save several design iterations without degrading performance by using only automatic learning. However, even better performance should be obtainable by combining the two approaches: automatic learning is ideal for optimising low level aspects of system behaviour such as determining appropriate confidence levels, confirmation strategy and management of initiative. Optimal treatment of these aspects depend on the user and error parameters encapsulated in the simulation tool and cannot be designed by hand in a principled way.

Methods such as manual design and automated planning, on the other hand, are useful for high level aspects of system design that are not influenced as directly by the simulation parameters. For instance, the order in which the system should request different pieces of information should not be decided by the reinforcement learner, as it is unlikely that the user model will contain information that is particularly relevant to the question. Instead, the issue should be resolved at a higher level without giving the learning algorithm a choice in the matter.

Such integration of automatic learning methods with more traditional approaches can be done by appropriate selection of the state space and action set that is presented to the reinforcement learner: by restricting the options available to the algorithm, it is allowed to focus on the design aspects for which its models contain useful information, without having to try to distinguish between high level strategies for which it is not suited. At the same time, this approach makes the problem more tractable so that a more detailed state space can be used. Closer investigation into the combination of automatic learning with other methods is an important topic for future research.

6.3.3 Training the user and error models using application independent data

One of the drawbacks of the simulation system is that training it requires data to have been obtained from an early prototype of the system being simulated. It would naturally be useful if good models could be created without such data. This possibility can be investigated by combining two alternative data sources. First, a general corpus of dialogues obtained from a wide range of applications can be used to obtain an application independent user model. Second, a set of utterances can be recorded in a context independent way to test the recognition grammars and obtain statistics of intention confusions and confidence measures.

These data sources do not require the prior existence of any versions of the dialogue system under development, so that they could be used at an early stage to bootstrap the design process. While models obtained from this data cannot be expected to be as accurate as those presented in this thesis, they should be accurate enough to save a number of early development cycles. Experiments should be conducted to evaluate the accuracy of such models.

6.3.4 Refining the user model

The user model of Chapter 4 is very simple, which has the advantage of reducing the required amount of training data. However, there are a number of possible refinements that might prove beneficial.

One such refinement that suggests itself is to use separate models for different subsets of the user population, e.g. a naive and an experienced user group. While the idea of basing the system design on an explicit model of user expertise is not new, no data-driven versions of such an approach seem to have been investigated yet. Once such models have been constructed, it would be possible to learn strategies that estimate and adapt to user expertise.

Another possibility is to increase the amount of dialogue history included in the user state description and model the dependence of user actions on this state explicitly. For instance, studies have found [7] that the probability of a user hanging up before completing a telephone conversation with a dialogue system is highly dependent on the recognition performance earlier in the dialogue. If this effect is modelled in the user model, it would improve the accuracy of the simulation and also allow the learning system to find strategies aimed at preventing such an outcome.

6.3.5 Expanding the state space

The problem of finding good state space representations for strategy learning is still far from solved. The state spaces used in the experiments of Chapter 5 can be expanded further, and other state variables can be tested to see whether they encapsulate useful information. In order to facilitate cross-domain porting of the results, the variables that are chosen should, like the state variables used in this thesis, be of the type that generalises easily to different applications.

Expanding the state space is especially likely to be useful after adding refinements to the user model. For instance, if the user model keeps track of the number of times a particular prompt has been repeated in the same conversation and models the user's reaction to such a phenomenon, it would be useful to also add such information to the state space. The learning algorithm would then be able to adjust the strategy accordingly.

6.3.6 Comparison with other learning approaches

Finally, it would be interesting to compare the approach used here with other learning approaches that have been studied. Experiments can be performed to compare the performance of the simulation-based and model-based approaches. This would give an idea of the relative accuracies of the models on which they are based.

The performance of these approaches can also be compared with the approximate POMDP approaches mentioned in Section 2.6.2.3. In order to do such a comparison, the state spaces used for the two approaches should not be the same: POMDP methods use state spaces which do not need to be fully observable, and are therefore not suitable for MDP methods. Similarly, a state space that works well for an MDP approach would probably not allow the advantages of the POMDP approach to come to the fore.

If it is found that approximate solutions for POMDPs can outperform MDP solutions, further experiments should be performed to investigate the combination of simulation-based methods with the POMDP framework.

Bibliography

[1] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmidt, and V. Zue, "GALAXY-II: A reference architecture for conversational system development," *Proc. ICSLP*, pp. 931–934, 1998.

[2] E. den Os, L. Boves, L. Lamel, and P. Baggia, "Overview of the ARISE project," *Proc. Eurospeech*, 1999.

[3] V. Zue, S. Seneff, J. R. Glass, J. Polifroni, C. Pao, T. J. Hazen, and L. Hetherington, "JUPITER: A telephone-based conversational interface for weather information," *IEEE Transactions on Speech and Audio Processing*, Vol. 8(1), 2000.

[4] M. Walker *et al.*, "DARPA Communicator dialog travel planning systems: The June 2000 data collection," *Proc. Eurospeech*, 2001.

[5] A. Rudnicky *et al.*, "Creating natural dialogs in the Carnegie Mellon Communicator system," *Proc. Eurospeech*, 1999.

[6] M. Rahim *et al.*, "Voice-IF: A mixed-initiative spoken dialogue system for AT&T Conference Services," *Proc. Eurospeech*, 2001.

[7] P. Durston *et al.*, "OASIS natural language call steering trial," *Proc. Eurospeech*, 2001.

[8] H. Asoh *et al.*, "A spoken dialogue system for a mobile office robot," *Proc. Eurospeech*, 1999.

[9] O. Lemon, A. Bracy, A. Gruenstein, and S. Peters, "The WITAS multi-modal dialogue system I," *Proc. Eurospeech*, 2001.

[10] J. Fry, S. Ginzton, S. Peters, B. Clark, and H. Pon-Barry, "Automated tutoring dialogues for training in shipboard damage control," *Proc. 2nd SIGdial Workshop on Discourse and Dialogue*, pp. 68–71, 2001.

[11] Y. Matsusaka, S. Fujie, and T. Kobayashi, "Modeling of conversational strategy for the robot participating in the group conversation," *Proc. Eurospeech*, 2001.

[12] V. Zue, "Conversational interfaces: Advances and challenges," *Proc. Eurospeech*, pp. KN9–KN18, 1997.

[13] J. Glass, "Challenges for spoken dialogue systems," *Proc. ASRU*, 1999.

[14] R. de Mori, "Recognising and using knowledge structures in dialog systems," *Proc. ASRU*, 1999.

[15] M. Lee, "Implicit goals in indirect replies," *ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, pp. 25–28, 1999.

[16] M. Core, J. Moore, and C. Zinn, "Initiative management for tutorial dialogue," *NAACL Workshop on Adaptation in Dialogue Systems*, 2001.

[17] P. Dalsgaard, C. Lee, P. Heisterkamp, and R. Cole, eds., *ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, June 1999.

[18] P. Cohen, "Dialogue modelling," in *Survey of the State of the Art in Human Language Technology* (R. Cole *et al.*, eds.), Oregon Graduate Institute of Technology, http://www.cse.ogi.edu/CSLU/HLTsurvey/, 1996.

[19] S. Bennacef, F. Néel, and H. Maynard, "An oral dialogue model based on speech acts categorization," *ESCA Workshop on Spoken Dialogue Systems - Theories and Applications*, pp. 237–240, 1995.

[20] M. Sadek, "Design considerations on dialogue systems: From theory to technology - the case of Artimis -," *ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, pp. 173–187, 1999.

[21] K. Wang, "A plan-based dialog system with probabilistic inferences," *Proc. ICSLP*, 2000.

[22] X. Wu, F. Zheng, and M. Xu, "Topic forest: A plan-based dialog management structure," *Proc. ICSLP*, 2000.

[23] P. Heisterkamp and S. McGlashan, "Units of dialogue management: an example," *Proc. ICSLP*, 1996.

[24] A. Gorin and G. Riccardi, "Spoken language variation over time and state in a natural spoken dialog system," *Proc. ICASSP*, 1999.

[25] F. Wessel and A. Baader, "Robust dialogue-state dependent language modelling using leaving-one-out," *Proc. ICASSP*, 1999.

[26] F. Wessel, A. Baader, and H. Ney, "A comparison of dialogue-state dependent language models," *ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, pp. 93–96, 1999.

[27] P. Baggia *et al.*, "Language modelling and spoken dialogue systems: the ARISE experience," *Proc. Eurospeech*, 1999.

[28] L. Bell, J. Boye, and J. Gustafson, "Real-time handling of fragmented utterances," *NAACL Workshop on Adaptation in Dialogue Systems*, pp. 2–8, 2001.

[29] A. Kellner, "Initial language models for spoken dialogue systems," *Proc. ICASSP*, 1998.

[30] S. Young and C. Proctor, "The design and implementation of dialogue control in voice operated database inquiry systems," *Computer Speech and Language*, Vol. 3, pp. 329–353, 1989.

[31] K. Komatani, K. Tanaka, H. Kashima, and T. Kawahara, "Domain-independent spoken dialogue platform using key-phrase spotting based on combined language model," *Proc. Eurospeech*, 2001.

[32] S. Shriver *et al.*, "Universalising speech: Notes from the USI project," *Proc. Eurospeech*, 2001.

[33] J. Sturm, E. den Os, and L. Boves, "Issues in spoken dialogue systems: Experiences with the Dutch ARISE system," *ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, pp. 1–4, 1999.

[34] E. Ammicht, A. Potamianos, and E. Fosler-Lussier, "Ambiguity representation and resolution in spoken dialogue systems," *Proc. Eurospeech*, 2001.

[35] M. Jansche, "Information extraction via heuristics for a movie showtime query system," *Proc. Eurospeech*, 2001.

[36] H. M. Meng, C. Wai, and R. Pieraccini, "The use of belief networks for mixed-initiative dialogue modeling," *Proc. ICSLP*, 2000.

[37] C. Wai, H. M. Meng, and R. Pieraccini, "Scalability and portability of a belief network-based dialog model for different application domains," *Proc. Human Language Technology*, pp. 309–314, 2001.

[38] S. F. Chan and H. M. Meng, "Interdependencies among dialog acts, task goals and discourse inheritence in mixed-initiative dialogs," *Proc. Human Language Technology*, 2002.

[39] H. M. Meng *et al.*, "ISIS: A learning system with combined interaction and delegation dialogs," *Proc. Eurospeech*, 2001.

[40] J. Bellegarda and K. Silverman, "Data-driven semantic inference for unconstrained desktop command and control," *Proc. Eurospeech*, 2001.

[41] J. Bellegarda, "Natural language interaction in goal-directed c&c: A comparative analysis of semantic inference," *Proc. Human Language Technology*, 2002.

[42] D. Goddeau, H. M. Meng, J. Polifroni, S. Seneff, and S. Busayapongchai, "A form-based dialogue manager for spoken language applications," *Proc. ICSLP*, pp. 701–704, 1996.

[43] J. Chu-Carrol, "Form-based reasoning for mixed-initiative dialogue management in information-query systems," *Proc. Eurospeech*, 1999.

[44] K. Papineni, S. Roukos, and R. Ward, "Free-flow dialogue management using forms," *Proc. Eurospeech*, 1999.

[45] D. Bobrow, R. Kaplan, M. Kay, D. Norman, H. Thompson, and T. Winograd, "GUS: A frame-driven dialog system," *Artificial Intelligence*, Vol. 8, pp. 155–173, 1977.

[46] C. Lavelle, M. de Calmès, and G. Pérennou, "Confirmation strategies to improve correction rates in a telephonic inquiry dialogue system," *Proc. Eurospeech*, 1999.

[47] R. San-Segundo, J. Montero, J. Ferreiros, R. Córdoba, and J. Pardo, "Designing confirmation mechanisms and error recover techniques in a railway information system for Spanish," *Proc. 2nd SIGdial Workshop on Discourse and Dialogue*, pp. 136–139, 2001.

[48] S. Singh, D. Litman, M. Kearns, and M. Walker, "Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system," *Journal of Artificial Intelligence Research*, Vol. 16, 2002.

[49] I. Lewin, "Limited enquiry negotiation dialogues," *Proc. Eurospeech*, 2001.

[50] G. Veldhuizen van Zandten, "User modelling in adaptive dialogue management," *Proc. Eurospeech*, 1999.

[51] D. Litman and S. Pan, "Designing and evaluating an adaptive spoken dialogue system," *User Modeling and User-Adapted Interaction*, 2001.

[52] M. Turunen and J. Hakulinen, "Agent-based error handling in spoken dialogue systems," *Proc. Eurospeech*, 2001.

[53] R. López-Cózar, A. Rubio, P. García, and J. Segura, "A new word-confidence threshold technique to enhance the performance of spoken dialogue systems," *Proc. Eurospeech*, 1999.

[54] A. Pargellis, H. J. Kuo, and C. Lee, "Automatic application generator matches user expectations to system capabilities," *ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, pp. 37–40, 1999.

[55] M. Rogati, M. Walker, and O. Rambow, "Training a sentence planner for spoken dialog: The impact of syntactic and planning features," *Proc. Eurospeech*, 2001.

[56] A. Ratnaparkhi, "Modeling informational novelty in a conversational system with a hybrid statistical and grammar-basd approach to surface natural language generation," *NAACL Workshop on Adaptation in Dialogue Systems*, pp. 49–56, 2001.

[57] K. Jokinen and G. Wilcock, "Confidence-based adaptivity in response generation fo a spoken dialogue system," *Proc. 2nd SIGdial Workshop on Discourse and Dialogue*, pp. 80–89, 2001.

[58] L. Bell and J. Gustafson, "Utterance types in the August dialogues," *ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, pp. 81–84, 1999.

[59] S. Whittaker and S. Stenton, "Wizard of Oz studies and the design of natural language systems," *Proc. EACL*, pp. 116–122, 1989.

[60] E. Ammicht, A. Gorin, and T. Alonso, "Knowledge collection for natural language spoken dialogue systems," *Proc. Eurospeech*, 1999.

[61] M. Hickey and P. Brittan, "Lessons from the development of a conversational interface," *Proc. Eurospeech*, 2001.

[62] G. Hanrieder, P. Heisterkamp, and T. Brey, "Fly with the EAGLES: Evaluation of the 'ACCeSS' spoken language dialogue system," *Proc. ICSLP*, 1998.

[63] M. A. Walker, D. J. Litman, C. A. Kamm, and A. Abella, "PARADISE: A general framework for evaluating spoken dialogue agents," *Proc. Conf. of the Association for Computational Linguistics*, pp. 271–280, 1997.

[64] M. A. Walker, D. J. Litman, C. A. Kamm, and A. Abella, "Evaluating spoken dialogue agents with PARADISE: Two case studies," *Computer Speech and Language*, Vol. 12(4), pp. 317–347, 1998.

[65] L. Devillers and H. Bonneau-Maynard, "Evaluation of dialog strategies for a tourist information retrieval system," *Proc. ICSLP*, 1998.

[66] L. B. Larsen, "Combining objective and subjective data in evaluation of spoken dialogues," *ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, pp. 89–92, 1999.

[67] M. Walker, J. Boland, and C. Kamm, "The utility of elapsed time as a usability metric for spoken dialogue systems," *Proc. ASRU*, 1999.

[68] S. Lee, E. Ammicht, E. Fosler-Lussier, H. Kuo, and A. Potamianos, "Spoken dialogue evaluation for the Bell Labs Communicator system," *Proc. Human Language Technology*, 2002.

[69] B.-S. Lin and L.-S. Lee, "Computer-aided analysis and design for spoken dialogue systems based on quantitative simulations," *IEEE Transactions on Speech and Audio Processing*, Vol. 9(5), pp. 534–548, 2001.

[70] Y. Niimi, T. Nishimoto, and Y. Kobayishi, "Analysis of interactive strategy to recover from misrecognition of utterances including multiple information items," *Proc. Eurospeech*, pp. 2251–2254, 1997.

[71] S. Furui and K. Yamaguchi, "Designing a multimodal dialogue system for information retrieval," *Proc. ICSLP*, 1998.

[72] Y. Niimi and T. Nishimoto, "Mathematical analysis of dialogue control strategies," *Proc. Eurospeech*, 1999.

[73] D. Louloudis, A. Tsopanoglou, N. Fakotakis, and G. Kokkinakis, "Mathematical modeling of spoken human-machine dialogues including erroneous confirmations," *Proc. Eurospeech*, 2001.

[74] R. Power, *A Computer Model of Conversation*. PhD thesis, University of Edinburgh, 1974.

[75] G. Houghton, *The Production of Language in Dialogue: A Computational Model*. PhD thesis, University of Sussex, 1986.

[76] E. Hovy, *Generating Natural Language under Pragmatic Constraints*. PhD thesis, Yale University, 1987.

[77] J. Carletta, *Risk-taking and Recovery in Task-Oriented Dialogue*. PhD thesis, University of Edinburgh, 1992.

[78] K. Hone and C. Baber, "Using a simulation method to predict the transaction time effects of applying alternative levels of constraint to user utterances within speech interactive dialogues," *ESCA Workshop on Spoken Dialogue Systems - Theories and Applications*, pp. 209–212, 1995.

[79] M. Araki and S. Doshita, "Automatic evaluation environment for spoken dialogue systems," in *Dialogue Processing in Spoken Language Systems* (E. Mayer, M. Mast, and S. LuperFoy, eds.), pp. 183–194, Springer, 1997.

[80] W. Eckert, E. Levin, and R. Pieraccini, "User modelling for spoken dialogue system evaluation," *Proc. IEEE ASR Workshop*, 1997.

[81] W. Eckert, E. Levin, and R. Pieraccini, "Automatic evaluation of spoken dialogue systems," Tech. Rep. TR98.9.1, AT&T Labs Research, 1998.

[82] K. Scheffler and S. Young, "Simulation of human-machine dialogues," Tech. Rep. CUED/F-INFENG/TR 355, Cambridge University Engineering Dept., 1999.

[83] K. Scheffler and S. Young, "Probabilistic simulation of human-machine dialogues," *Proc. ICASSP*, pp. 1217–1220, 2000.

[84] K. Scheffler and S. Young, "Corpus-based dialogue simulation for automatic strategy learning and evaluation," *Proc. NAACL Workshop on Adaptation in Dialogue*, 2001.

[85] B.-S. Lin and L.-S. Lee, "Fundamental performance analysis for spoken dialogue systems based on a quantitative simulation approach," *Proc. ICASSP*, pp. II–1221–1224, 2000.

[86] M. Walker, D. Hindle, J. Fromer, G. Di Fabbrizio, and C. Mestel, "Evaluating competing agent strategies for a voice email agent," *Proc. Eurospeech*, pp. 2219–2222, 1997.

[87] J. Relaño Gil *et al.*, "Flexible mixed-initiative dialogue for telephone services," *Proc. Eurospeech*, 1999.

[88] M. Walker, *Informational Redundancy and Resource Bounds in Dialogue*. PhD thesis, University of Pennsylvania, 1993.

[89] E. Levin and R. Pieraccini, "A stochastic model of computer-human interaction for learning dialogue strategies," *Proc. Eurospeech*, pp. 1883–1886, 1997.

[90] E. Levin, R. Pieraccini, and W. Eckert, "Using Markov decision process for learning dialogue strategies," *Proc. ICASSP*, 1998.

[91] E. Levin, R. Pieraccini, and W. Eckert, "A stochastic model of human-machine interaction for learning dialogue strategies," *IEEE Transactions on Speech and Audio Processing*, Vol. 8(1), pp. 11–23, 2000.

[92] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts; London, England: MIT Press, 1998.

[93] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, No. 4, pp. 237–285, 1996.

[94] S. Young, "Probabilistic methods in spoken dialogue systems," *Philosophical Transactions of the Royal Society (Series A)*, Vol. 358, pp. 1389–1402, 2000.

[95] D. Goddeau and J. Pineau, "Fast reinforcement learning of dialog strategies," *Proc. ICASSP*, 2000.

[96] K. Scheffler and S. Young, "Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning," *Proc. Human Language Technology*, 2002.

[97] R. Lecœuche, "Learning optimal dialogue management rules by using reinforcement learning and inductive logic programming," *Proc. NAACL*, pp. 216–222, 2001.

[98] M. Walker, J. Fromer, and S. Narayanan, "Learning optimal dialogue strategies: a case study of a spoken dialogue agent for email," *Proc. Association for Computational Linguistics*, pp. 1345–1352, 1998.

[99] M. Walker, "An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email," *Journal of Artificial Intelligence Research*, Vol. 12, pp. 387–416, 2000.

[100] S. Singh, M. Kearns, D. Litman, and M. Walker, "Reinforcement learning for spoken dialogue systems," *Proc. NIPS*, 1999.

[101] S. Singh, M. Kearns, D. Litman, and M. Walker, "Empirical evaluation of a reinforcement learning spoken dialogue system," *Proc. AAAI*, 2000.

[102] D. Litman, M. Kearns, S. Singh, and M. Walker, "Automatic optimization of dialogue management," *Proc. COLING*, 2000.

[103] N. Roy, J. Pineau, and S. Thrun, "Spoken dialogue management using probabilistic reasoning," *Proc. Association for Computational Linguistics*, 2000.

[104] J. Pineau, N. Roy, and S. Thrun, "A hierarchical approach to POMDP planning and execution," *Workshop on hierarchy and memory in reinforcement learning (ICML)*, June 2001.

[105] B. Zhang, Q. Cai, J. Mao, and B. Guo, "Planning and acting under uncertainty: A new model for spoken dialogue systems," *Uncertainty in Artificial Intelligence: Proceedings of the 17th Conference*, pp. 572–579, 2001.

[106] B. Zhang, Q. Cai, J. Mao, E. Chang, and B. Guo, "Spoken dialogue management as planning and acting under uncertainty," *Proc. Eurospeech*, 2001.

[107] M. Denecke, "Informational characterization of dialogue states," *Proc. ICSLP*, 2000.

[108] C. Doran, J. Aberdeen, L. Damianos, and L. Hirschman, "Comparing several aspects of human-computer and human-human dialogues," *Proc. 2nd SIGdial Workshop on Discourse and Dialogue*, pp. 48–57, 2001.

[109] W. Eckert, E. Nöth, H. Niemann, and E. Schukat-Talamazzini, "Real users behave weird - experiences made collecting large human-machine-dialog corpora," *ESCA Workshop on Spoken Dialogue Systems - Theories and Applications*, pp. 193–196, 1995.

[110] C. Watkins, *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.

[111] M. D. Pendrith and M. J. McGarity, "An analysis of direct reinforcement learning in non-Markovian domains," *The Fifteenth International Conference on Machine Learning*, July 1998.

[112] J. Loch and S. Singh, "Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes," *The 15th International Conference on Machine Learning*, July 1998.