Probabilistic Methods in Spoken Dialogue Systems

By Steve Young

Cambridge University Engineering Department
Trumpington Street, Cambridge CB2 1PZ
email: `sjy@eng.cam.ac.uk`

This paper presents a probabilistic framework for modelling spoken dialogue systems. On the assumption that the overall system behaviour can be represented as a Markov Decision Process, the optimisation of dialogue management strategy using reinforcement learning is reviewed. Examples of learning behaviour are presented for both dynamic programming and sampling methods, but the latter is preferred. The paper concludes by noting the importance of user simulation models for the practical application of these techniques and the need for developing methods of mapping system features in order to achieve sufficiently compact state spaces.

Keywords: speech interfaces dialogue systems reinforcement learning

1. Introduction

Speech technology is maturing rapidly and attention is switching to the problems of using this technology in real applications, especially applications which allow a human to use voice to interact directly with a computer-based information or control system. Apart from the simplest of cases, such systems involve a sequence of interactions between a person and a machine. They therefore involve dialogue and discourse management issues in addition to those associated with prompting the user and interpreting the responses. These systems are often referred to as Spoken Dialogue Systems (SDS).

Existing SDS are diverse in nature and there are many papers describing them e.g. [Young and Proctor, 1989, Aust et al., 1995, Sadek et al., 1995, Pieraccini et al., 1997, Zue, 1997]. There are also several books providing more extensive treatment of the issues [Smith and Hipp, 1994, Bernsen et al., 1998, De Mori, 1998].

The main components of a typical SDS are illustrated in Fig 1. Overall operation is orchestrated by a dialogue manager whose function is to exchange information with the user and thereby access and update the information in the database. User interaction consists of a sequence of question/answer cycles where each question is designed to illicit some specific information. The user's response is processed by a speech recogniser and its output is converted by an interpreter into a semantic representation. Based on this new input, the dialogue manager updates its internal state and plans its next action. This continues until the user's information need is satisfied and the interaction is terminated.

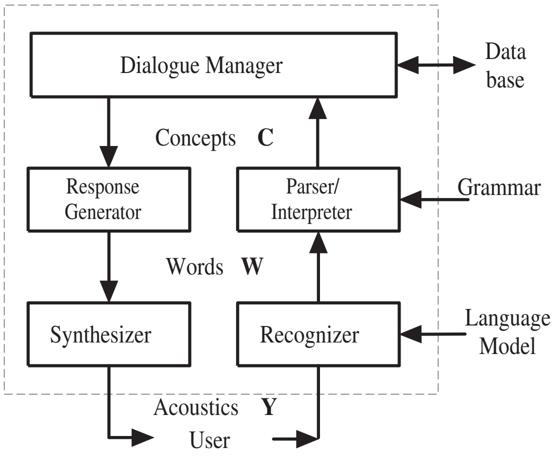


Figure 1. Architecture of a Spoken Dialogue System

In terms of data flow, the user's input is an acoustic waveform **Y** which is converted by the recogniser into a sequence of words **W**. The parsing and interpreting stage converts the word sequence into a concept sequence **C** which typically consists of tagged entities arranged in some predicate-argument structure. The recognition process makes use of a language model to constrain the search space of possible word sequences, and the parser makes use of a grammar to define the phrase structure of the input. Output to the user follows the same pattern in reverse, concepts are converted to words which are then converted to acoustic waveforms.

The key issues in designing an SDS are how to specify and control the dialogue flow; how to constrain the recogniser to work within a limited domain; how to interpret the recognition output; and how to generate contextually appropriate responses to the user. The design criteria which motivate the solutions to these issues are many and varied but the key top-level goal is to produce a system which allows a user to complete their required tasks quickly and accurately.

The lower level components of an SDS typically make heavy use of probabilistic approaches both in their design and run-time operation. For example, the speech recognizer often uses hidden Markov models to represent the basic sounds of the language and N-gram language models to represent the patterns of expected word sequences. Both the acoustic models and the language models are estimated from large training corpora during the construction process. At run-time Viterbi decoding is used to find the sequence of acoustic models and words which are most likely to have generated the observed acoustic input [Young, 1996].

The use of probabilistic methods in speech recognition has proved to be essential to obtaining robust operation. By training the models on large corpora which are representative of the intended user population, the full range of variability in speakers, accents, moods, etc can be accommodated. However, current SDS are typically designed at the higher levels using a combination of heuristics and iterative refinement. The aim of this paper is to review and promote discusion of the areas in which the probabilistic approaches used inside the recognition and parsing components can be extended to the higher levels of system design. In particular, the paper addresses the following issues:

- How can dialogue management strategies by optimised?
- How can robust dialogue management strategies be learnt automatically?
- How can expected performance be estimated for a given dialog design?

The type of SDS considered here is limited to systems in which the user has a clear goal and the dialogue is managed using a strategy which is fixed in advance. As will become clear later, a key assumption is that the system memory and dialogue history can be encoded within a single state variable.

The remainder of the paper is organised as follows. Section 2 outlines a general probabilistic framework for modelling SDS. Section 3 then discusses dialogue management in terms of Markov Decision Processes (MDP's). Sections 4 and 5 describe the two basic approaches to parameter estimation and optimisation within the MDP framework. Section 6 describes how a user simulation model can be used to aid development and evaluation. Finally, in section 7, conclusions are presented and possible areas of future work are discussed.

2. A Probabilistic Framework for SDS

The primary functional units of an SDS are shown in Fig 2 which is an alternative view of the architecture diagram shown earlier in Fig 1. The system is controlled by a dialog manager which generates a sequence of *actions* a_t dependent upon the system state s_t. The goal of the dialog manager is to change the system from some initial uninformed state to a sufficiently informed state that the user's information need can be satisfied. The actions are primarily questions to the user although they can also result in accesses to a database. Questions to the user result in acoustic responses $\mathbf{y}_t$ which are corrupted by noise $\mathbf{n}_t$ before being input to the *speech understanding system (SUS)*. The concepts $\mathbf{c}_t$ output from the SUS cause the system memory to be updated leading to a new dialog state s_{t+1}. The relationship between the dialog state and the concepts output by the SUS is determined by the specific task. Also, each step taken by the dialogue manager results in the generation of a *reward* r_{t+1}.

The rewards generated at each dialogue step represent the system design objectives and the overall goal of the system design process is to maximise the total reward $R = \sum_{t=1}^{T} r_t$. Typically, each user interaction will incur a small negative reward and successfully meeting the user's information need will generate a large positive reward [Levin and Pieraccini, 1997]. The complete sequence of user interactions leading from the initial state to the final state is called a *dialogue transaction* and the primary goal is to satisfy the user's requirements whilst minimising the transaction time. If there are other dialog design criteria such as minimising the number of corrections, or minimising the number of database accesses, then these can be incorporated into the reward function accordingly.

Within this framework, the joint distribution for the state, action, speech and concept sequence can be decomposed as

$$P(s_{t+1}, a_t, \mathbf{y}_t, \mathbf{c}_t | s_t, \mathbf{n}_t) = \underbrace{P(s_{t+1}|\mathbf{c}_t, s_t)}_{\text{Task Model}} \underbrace{P(a_t|s_t)}_{\text{DM}} \underbrace{P(\mathbf{y}_t|a_t, s_t)}_{\text{User}} \underbrace{P(\mathbf{c}_t|\mathbf{y}_t, a_t, s_t, \mathbf{n}_t)}_{\text{SUS}} \tag{2.1}$$

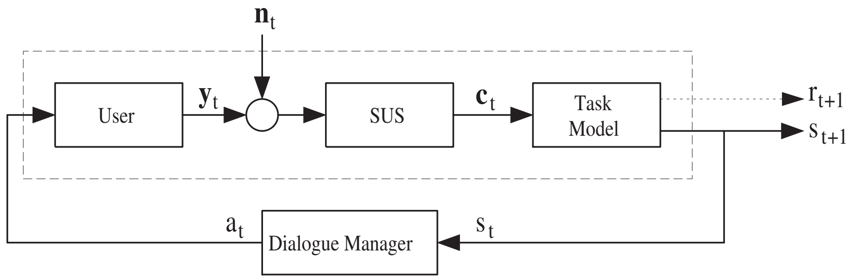


Figure 2. Block Diagram of a Spoken Dialogue System

The dialogue manager term $P(a_t|s_t)$ represents the dialogue control strategy i.e. in each state of the dialogue, what is the best action to take next? The user term $P(\mathbf{y}_t|a_t, s_t)$ represents the response of each user to receiving a specific query action in a given state. This term will either represent a model of the typical user or the statistics of a real user population. Finally, the SUS term $P(\mathbf{c}_t|\mathbf{y}_t, a_t, s_t, \mathbf{n}_t)$ represents the speech recognition and understanding processes. The conditioning terms indicate that the interpretation of a user input $\mathbf{y}_t$ can depend on the query action, the dialogue state and the ambient noise conditions.

The focus of this paper is on the first three terms of equation 2.1 but before moving on, the current state of relevant work on the SUS component should be noted. The final term in equation 2.1 can be expanded (ignoring the time step index) as

$$P(\mathbf{c}|\mathbf{y}, a, s, \mathbf{n}) \approx max_{\mathbf{w}} P(\mathbf{c}|\mathbf{w}, a, s) P(\mathbf{w}|\mathbf{y}, a, s, \mathbf{n}) \tag{2.2}$$

The second term defines the conventional speech recognition problem which, ignoring the noise term, can be written as

$$P(\mathbf{w}|\mathbf{y}, a, s) = P(\mathbf{y}|\mathbf{w}) P(\mathbf{w}|a, s) / P(\mathbf{y}|a, s) \tag{2.3}$$

where $P(\mathbf{y}|\mathbf{w})$ is the acoustic likelihood of the observed speech given the word sequence $\mathbf{w}$ and $P(\mathbf{w}|a, s)$ is the prior probability of the word sequence $\mathbf{w}$ given the action and dialogue state. There is a substantial literature on solutions to this modelling problem [Young, 1996, provides a review], however, the interesting aspect from the point of view of SDS is the explicit inclusion of the dependence of $P(\mathbf{w})$ on the action a and dialogue state s. This is the crucial feature which makes complex interactive SDS feasible. The language model used at each question/answer step can be highly context dependent. For example, it can be tightly constrained to a specific set of possible input utterances as in [Young et al., 1991] or it can be a stochastic context dependent n-gram as in [Andry, 1992, Niedermaier, 1992, Drenth and Rüber, 1997].

The mapping of words to concepts as represented by the first term in equation 2.2 is typically done by mapping phrases to semantic tags or predicates. This mapping can be done manually using explicit grammars as in [Ward, 1991, Seneff et al., 1992] or using n-grams as in [Pieraccini et al., 1991, Pieraccini and Levin, 1992, Miller and Bobrow, 1994]. These approaches can also be combined as in [Meteer and Rohlicek, 1993, Lloyd-Thomas et al., 1995]. Most current approaches identify the salient phrases to extract and map manually. However, if the task is simple enough and there is a large body of data to learn from, it is possible to identify the salient phrases automatically [Gorin et al., 1991, Gorin et al., 1997, Arai et al., 1999].

3. Markov Decision Processes

The remainder of this paper focusses on the key issue of dialogue management. From the point of view of the dialogue manager, the components within the dotted box in figure 2 can be regarded as a single system driven by the input actions a_t. If each dialogue action and state change are assumed to depend only on the current state, then the system can be modelled as a Markov Decision Process (MDP)[Levin and Pieraccini, 1997, Levin et al., 1998].

The key equations governing the behaviour of an MDP are:

- transition function $T(s', a, s) = P(s_{t+1} = s' | a_t = a, s_t = s)$
- policy matrix $\pi(s, a) = P(a_t = a | s_t = s)$
- expected reward $R(s', a, s) = \mathcal{E}(r_{t+1} | s_{t+1} = s', a_t = a, s_t = s)$

The policy matrix represents the dialogue management strategy and the reward represents the objective function. Assuming that a dialogue transaction consists of a sequence of states $s_0, s_1, \ldots, s_T$, then the total expected reward for that transaction is just $\mathcal{R} = \sum_{t=1}^{T} R(s_{t+1}, a_t, s_t)$ and the goal is usually to find a policy which maximises this. Markov decision process optimisation is a well-studied topic [Kaelbling et al., 1996, Sutton and Barto, 1998, for example] and only the key points will be summarised here.

The expected value of the reward can be computed recursively by introducing a *value function* $V^\pi(s)$ which is the expected reward from state s to terminal state s_T given policy π.

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} T(s', a, s)[R(s', a, s) + V^\pi(s')] \tag{3.1}$$

$$= \sum_a \pi(s, a) Q^\pi(s, a) \tag{3.2}$$

where $Q(s, a)$ gives the expected reward if action a is taken from state s.

The optimal dialogue policy $\pi^*(s)$ is the deterministic policy which gives the maximum value function $V^*(s)$ such that

$$V^*(s) = \max_\pi V^\pi(s) \quad \forall s \in S \tag{3.3}$$

where S is the set of all states. The optimal value function can be found by solving

$$V^*(s) = \max_a \sum_{s'} T(s', a, s)[R(s', a, s) + V^*(s')] \tag{3.4}$$

Similarly, the optimal Q function can be found by solving

$$Q^*(s, a) = \sum_{s'} T(s', a, s)[R(s', a, s) + \max_{a'} Q^*(s', a')] \tag{3.5}$$

and then the required optimal dialogue policy is given by

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{3.6}$$

These are the basic equations which govern MDP's and which can be used as the basis for dialogue modelling and optimisation. Equations 3.4 and 3.5 are non-linear and have no closed form solution. Therefore iterative methods are used. Furthermore, any implementation of reinforcement learning for realistic dialogue systems must be able to handle problems of accurately estimating the state transition function and dealing with very large state spaces. Taking these issues into account, two distinct approaches can be identifed: dynamic programming and sampling, and these are discussed in more detail below. A further issue is the source of data used to estimate parameters. Ideally, the data would be gathered from interactions with real users but this is rarely practical, at least during the initial design phases. An alternative approach is to build a model which simulates users. This topic is also discussed below.

The two primary performance metrics for a dialogue management strategy are the *probability of success* $P_{\rm succ}$ and the *expected transaction time* $\mathcal{E}[d]$. Once the system transition function and dialogue policy have been learnt, these can be easily computed. If the dialogue policy is deterministic, the state transition function $T(s,a,s')$ can be mapped into an $|S| \times |S|$ transition matrix $T[s,s']$. Otherwise, the state matrix must be expanded for each (s,a) pair. Given an initial, state vector $\mathbf{p}_0$, the state vector at time t, and hence the probability of occupying each terminal state, is given by

$$\mathbf{p}_t = T^t \mathbf{p}_0 \tag{3.7}$$

As is well known, this can be evaluated efficiently by setting $T = VDV^{-1}$ where D is a diagonal matrix of eigenvalues and V is a matrix whose columns are the corresponding eigenvectors. Then

$$\mathbf{p}_t = VD^tV^{-1}\mathbf{p}_0 \tag{3.8}$$

and $P_{\rm succ} = p_t(n), t \to \infty$ where n is the state index of the terminal *success* state. The expected transaction time can be computed in the obvious way by computing the expected value of t in moving from state 1 to a terminal state i.e. $\sum_{t=1}^{\infty} \sum_{m \in \mathcal{T}} p_t(m)t$, where $\mathcal{T}$ is the set of all terminal states†

4. Dynamic Programming based Policy Optimisation

The basic idea of Dynamic Programming (DP) based reinforcement learning methods is to scan the state space in order to recursively estimate the value function for a given policy. Once the value function has been computed, the policy itself can be updated and the whole process repeated. This procedure is guaranteed to maximise the value function and converge to the optimal policy.

There are several variants of this approach but the simplest and most effective is *value iteration* whereby the policy is effectively updated at every step within

† To compute $P_{\rm succ}$, the terminal success state must be absorbing i.e. have a self-loop of probability 1. To compute expected transaction times, the terminal states must be non-absorbing.

the value function estimation. The value update equation follows immediately from equation 3.4 above. For each $s \in S$ until convergence

$$V(s) \leftarrow \max_a \sum_{s'} T(s', a, s)[R(s', a, s) + V(s')] \tag{4.1}$$

This effectively interleaves the value function estimation and policy updates by always choosing the policy which is locally maximum. If the best action for each state is recorded in $\pi(s)$, then on convergence $\pi(s)$ will be the optimal policy.

As noted earlier, this approach is limited by the size of the state space and the difficulty of estimating the state transition function. These problems can be bypassed to some degree by maintaining a reduced state representation and using global recognition/understanding statistics.

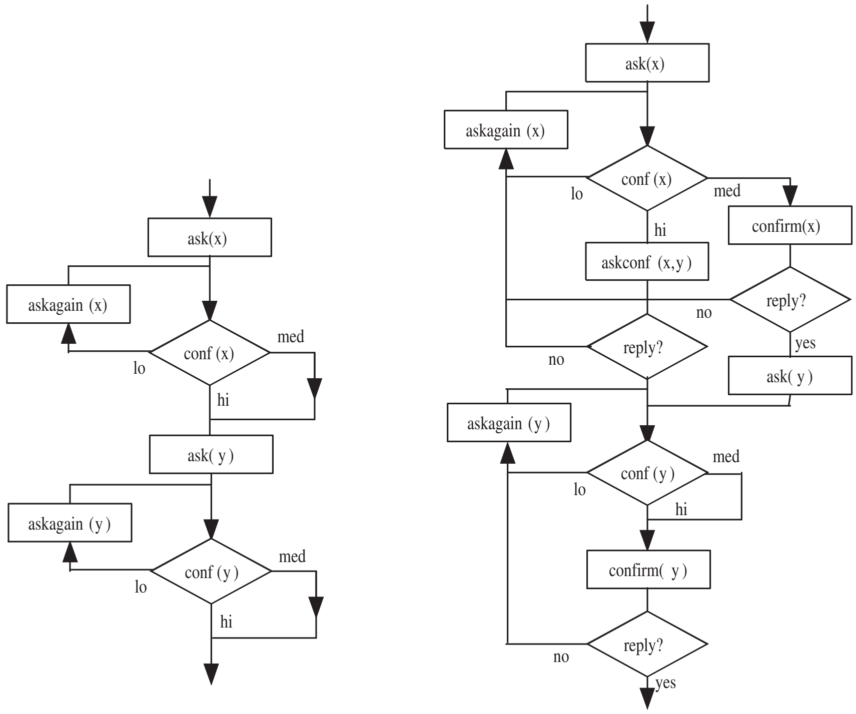
As an example, many dialog "form-filling" applications can be reduced to the problem of finding a set of values and then retrieving some required information from a database. The system state in this case can be represented by the state of each variable where each variable is either unknown (uk), known with some degree of confidence, say low(lo), medium (md) or high (hi)), confirmed and correct (cc), or confirmed but false (cf). Thus, for n variables, in this example there would be 6^n dialogue states.

Suppose that a dialogue management policy is required based on the assumption that in any state one of the following 5 actions are possible:

ask(x)	ask the user for the value of variable x
confirm(x)	ask the user to confirm the value of x
askconf(x,y)	ask for the value of y whilst implicitly confirming x
askagain(x)	ask again for the value of x
accept(x)	accept the value of x without confirming it

Further assume that the performance of the SUS component can be idealised by the global statistics: $P(l|a)$ and $P(err|l)$ where l is the confidence level. Thus, for example, in response to *ask(x)*, the state of variable x would change from uk to md with probability $P(md|ask)$. A subsequent *confirm(x)* action would then change the state of variable x from md to cc with probability $P(corr|md) \sum_l P(corr|l)P(l|confirm)$ where $P(corr|l) = 1 - P(err|l)$ and the summation represents the probability of the user being correctly recognised as saying "yes". Similarly, the state of x would change from md to lo representing the case where the user does not confirm the value of x with probablity $P(err|md) \sum_l P(corr|l)P(l|confirm) + P(corr|md) \sum_l P(err|l)P(l|confirm)$. In this case, the first term corresponds to the case where the user is correctly recognised as saying "No" and the second term corresponds to the case where the user is misrecognised as saying "No" when x was actually correct.

Using this type of model, Figure 3(a) shows the optimal dialogue strategy learnt for a two variable problem using a near perfect recogniser. In this case, no confirmation is attempted. If the recogniser confidence is low, it asks again, otherwise it assumes that the recognized value is correct. Figure 3(b) shows the strategy learnt for a recogniser with characteristics typical of an imperfect but good system. In this case, both variables are confirmed. However, the first variable is confirmed implicitly if the confidence is high in order to potentially save a question/answer cycle. If



(a) Near-perfect recognition (b) Imperfect recognition

Figure 3. Dialogue Policies Learnt for Near Perfect and Imperfect Recognisers

the recognition degrades further, then the implicit *askconf(x)* action is replaced by an explicit confirm action.

The reward function used for these examples was -1 for each question-answer cycle and +100 for a successful conclusion. The final values of $V(0)$ for the perfect and good recognisers were 98 and 82.5, respectively. The probability of success was 99% and 83%, and the expected number of dialogue steps was 2 and 4.94, respectively.

5. Sampling based Policy Optimisation

Whereas DP methods aim to explore the complete state space, sampling methods focus on state sequences corresponding to optimal and near optimal dialogues. They can therefore be used when a complete model of the user and/or the SUS component is unavailable.

A simple and effective sampling approach is to use the Monte Carlo technique on whole dialogue transactions. In this method, a sequence of dialogue transactions is generated using a stochastic policy $\pi(s, a)$. For each pair (s, a) appearing in the transaction, the corresponding estimate of $Q(s, a)$ is updated by sample average

$$Q(s, a) \leftarrow ([N(s, a) - 1]Q(s, a) + R(s, a))/N(s, a) \tag{5.1}$$

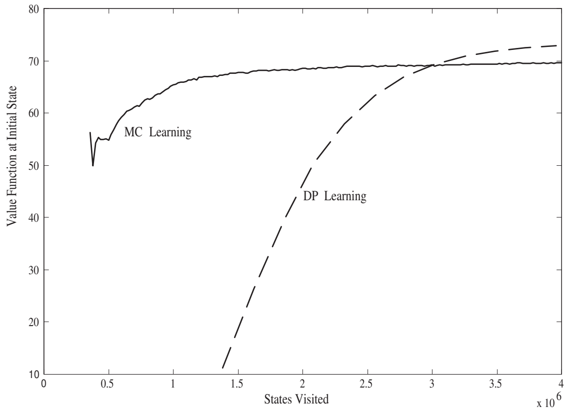


Figure 4. Value function versus number of states visited for learning using Dynamic Programming (DP) and using Monte Carlo Sampling (MC)

where $R(s, a)$ is the actual reward obtained from (s, a) and $N(s, a)$ is the number of times that (s, a) has been visited. In this way, the Q function asymptotically approaches its expected value over a continuing sequence of transactions and the use of a stochastic policy ensures that the relevant areas of the state space can be fully explored.

The most common form of stochastic policy is the ϵ-soft policy. For each pair (s, a), let $a^* = \arg\max_a Q(s, a)$ then

$$\pi(s, a) = \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \epsilon/|\mathcal{A}(s)| & \text{otherwise} \end{cases} \tag{5.2}$$

where $\mathcal{A}(s)$ is the set of all actions possible from state s. This policy is designed to mostly follow the locally optimal policy but to occasionally choose with probability $\epsilon/|\mathcal{A}(s)|$ a non-optimal action. As $\epsilon \to 0$, then the soft policy hardens to the optimal policy.

To illustrate the relative performance of the DP and MC algorithms described in this and the previous section, the value function at the initial state can be plotted as a function of the number of states visited. Fig 4 plots these for a 3 variable dialogue of the form described earlier. As can be seen, the behaviour of the two algorithms is very different. In DP, the terminal reward propagates slowly back towards the initial state and the value function is approached smoothly from below whilst it is simultaneously optimised by adjusting the policy. In MC, the terminal reward is propagated back to the initial state after just one transaction, however, it will be a poor estimate since only one of the many possible state sequences will have been visited. DP is relatively slow compared to MC because it takes account of all possible state sequences whereas MC only considers state sequences with a high probability. However, DP is guaranteed to find the optimal policy whereas the convergence properties of MC depend on the "hardening" schedule of the ϵ-soft policy. In practice, for the example here, MC finds the optimal policy typically two to three times faster than DP.

For long and complex dialogues, the basic MC technique may be slow to converge because parameter updates only occur on the completion of each transaction when the final rewards are known. An alternative sampling technique is *Temporal Difference (TD) learning* in which the Q function is updated after each action by comparing the actual 1-step reward with the reward predicted by the Q function. More specifically, at (s, a), the action a is applied and the reward r and the new state s' is observed. Then the next action a' is selected using an ϵ-soft policy derived from Q. This is a one-step lookahead which allows Q to be updated by

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + Q(s',a') - Q(s,a)] \tag{5.3}$$

where α controls the learning rate. After updating Q using equation 5.3, the process repeats from (s', a') and so on until the end of the transaction.

As a final comment on sampling versus DP methods, it should be noted that sampling methods do not require full knowledge of the state space, and furthermore, they can be adapted to track systems where the transition function is non-stationary. Thus, in general, sampling methods are preferred over dynamic programming methods.

6. User Simulation

As noted earlier, one of the disadvantages of the dynamic programming approach to dialogue policy optimisation is that a complete knowledge of the system transition function is required. Sampling approaches on the other hand can be applied to practical systems by learning directly from each dialogue transaction as it evolves. By using a policy which occasionally strays away from the optimal, new and perhaps better policies can be found.

In practice, it takes many dialogue transactions to derive an optimal policy and on-line learning from real users may be impractical. An alternative is to use a model which simulates typical user behaviour [Eckert et al., 1997]. This model can then be interfaced to the dialogue manager and SUS component to generate as many dialogue transactions as required.

The simulated user component can be interfaced at the speech level, the word level or the tag level. The tag level focusses on the key informational items in an utterance (e.g. *flight-info-request, leave-place, arrival-time, amount, etc*) and it corresponds roughly to the user's intentions. For dialogue modelling, interfacing at the intention level is the most convenient since it allows the effects of recognition and understanding errors to be modelled whilst avoiding the intricacies of natural language generation.

An intention level simulated user component will typically consist of a probabilistic finite state tag generator. A tag sequence is generated by traversing through a sequence of *choice points*. These choices reflect the range of possible user inputs constrained by some overall goal. The history of choices must be recorded so that the user behaviour is rational and does not, for example, unnecessarily respecify information which has already been given. As well as choices concerned with differing intentions and varying levels of mixed initiative, choices also include meta-level options such as those associated with the user being confused by where he or she is in the dialogue and by what is expected at that point. The probabilities governing typical user choices can be estimated from prototype trials or from *Wizard-of-Oz* experiments.

The generated tag sequence is filtered by an error module which simulates recognition and SUS component errors. It should also model user errors such as saying something which is not intended. The required probabilities for this type of error model can be determined from data collected from deployed systems by comparing reference transcriptions with recognition output aligned at the tag level.

A more detailed description of a probabilistic simulated user component of this form is given by [Scheffler and Young, 1999].

7. Conclusions

This paper has described a probabilistic framework for modelling human-computer dialogues. The framework assumes that spoken dialogue systems can be modelled as Markov Decision Processes and are therefore amenable to reinforcement learning techniques. Given either an explicit state transition function, a user simulation or an operational system, a reward value function can be estimated and an optimal dialogue strategy determined. Once that dialogue strategy is fixed, a variety of operational characteristics can be estimated.

This dialogue modelling approach has application in a number of areas. For example,

- estimating the probability of success and expected transaction time
- optimising an existing dialogue strategy
- automatically learning an optimal dialogue strategy
- optimising the use of recognition confidence measures
- on-line adaptation of dialogue strategy when conditions are time-varying

It can therefore be usefully applied within dialogue design tools, as part of the analysis of existing dialogue systems, and within the adaptation mechanisms of operational systems.

These approaches to probabilistic dialogue modelling are still at a very early stage and it is not clear how generally applicable they will be in practice. Clearly a crucial issue is the apparent dependence on a single global state variable and the Markov assumption. Reinforcement learning depends crucially on the estimation of a state value function $V(s)$ which allows the alternative actions available at each state to be compared. All of the methods reviewed in this paper have assumed that $V(s)$ is a look-up table and that the states are distinct. In practical systems, the state-space will be very large and possibly unknown. Distinct states arise from the combination of system characteristics such as the status of variables and their values, the recognition performance, prior information such as defaults, historical information such as the number of times a variable has been requested and so on. Thus, to make progress it will be necessary to find good ways of combining these state features and mapping them to form a compact and manageable state-space. Possible lines of attack here include using Neural Networks to automatically learn an appropriate mapping or Bayesian networks to allow the differing types of user and system level information to be combined [Pulman, 1997].

Other important research areas include finding the best reinforcement learning methods to use for dialogue systems and building effective user simulation models. In the much longer term, the various system components such as the acoustic language models, grammars, confidence levels and dialogue manager should be jointly optimised to maximise the desired system performance. However, this ideal may take some time to achieve!

Acknowledgements

The focus of this paper owes much to the work of Esther Levin and Roberto Pieraccini. Karen Sparck Jones, Gerald Gazdar and Konrad Scheffler provided helpful feedback on an early draft.

References

[Andry, 1992] Andry, F. (1992). Static and dynamic predictions: a method to improve speech understanding in cooperative dialogues. In *Proc International Conference on Spoken Language Processing*, pages 639–642, Banff, Canada.

[Arai et al., 1999] Arai, K., Wright, J., Riccardi, G., and Gorin, A. (1999). Grammar fragment aquisition using syntactic and semantic clustering. *Speech Communication*, 27(1):43–62.

[Aust et al., 1995] Aust, H., Oerder, M., Seide, F., and Steinbiss, V. (1995). A spoken language inquiry system for automatic train timetable information. *Philips Journal of Research*, 49:399–418.

[Bernsen et al., 1998] Bernsen, N., Dybkjaer, H., and Dybkjaer, L. (1998). *Designing Interactive Speech Systems*. Springer-Verlag, London.

[De Mori, 1998] De Mori, R. (1998). *Spoken Dialogues with Computers*. Signal Processing and its Applications. Academic Press.

[Drenth and Rüber, 1997] Drenth, E. and Rüber, B. (1997). Context-dependent probability adaptation in speech understanding. *Computer Speech and Language*, 11(3):225–252.

[Eckert et al., 1997] Eckert, W., Levin, E., and Pieraccini, R. (1997). User modelling for spoken dialogue system evaluation. In *Proc IEEE ASR Workshop*, Santa Barbara.

[Gorin et al., 1991] Gorin, A., Levinson, S., Gertner, A., and Golman, E. (1991). Adaptive acquisition of language. *Computer Speech and Language*, 5(2):101–132.

[Gorin et al., 1997] Gorin, A., Riccardi, G., and Wright, J. (1997). How may I help you? *Speech Communication*, 23(1-2):113–127.

[Kaelbling et al., 1996] Kaelbling, L., Littman, M., and Moore, A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:238–285.

[Levin and Pieraccini, 1997] Levin, E. and Pieraccini, R. (1997). A stochastic model of computer-human interaction for learning dialogue strategies. In *Proc Eurospeech*, pages 1883–1886, Rhodes,Greece.

[Levin et al., 1998] Levin, E., Pieraccini, R., and Eckert, W. (1998). Using markov decision processes for learning dialogue strategies. In *Proc Int Conf Acoustics, Speech and Signal Processing*, Seattle,USA.

[Lloyd-Thomas et al., 1995] Lloyd-Thomas, H., Wright, J., and Jones, G. (1995). An integrated grammar/bigram language model using path scores. In *Proc ICASSP*, volume 1, pages 173–176, Detroit.

[Meteer and Rohlicek, 1993] Meteer, M. and Rohlicek, R. (1993). Statistical language modelling combining n-gram and context free grammars. In *Proc ICASSP*, volume 2, pages 37–40, Minneapolis. IEEE Press.

[Miller and Bobrow, 1994] Miller, S. and Bobrow, R. (1994). Statistical language processing using hidden understanding models. In *Proc Human Language Technology Workshop*, pages 278–282, Plainsboro, NJ. ARPA.

[Niedermaier, 1992] Niedermaier, G. (1992). Linguistic modelling in the context of oral dialogue. In *Proc International Conference on Spoken Language Processing*, pages 635–638, Banff, Canada.

[Pieraccini and Levin, 1992] Pieraccini, R. and Levin, E. (1992). Stochastic representation of semantic structure for speech understanding. *Speech Communication*, 11(2):283–288.

[Pieraccini et al., 1997] Pieraccini, R., Levin, E., and Eckert, W. (1997). Amica: the at&t mixed initiative conversational architecture. In *Proc Eurospeech*, pages 1875–1878, Rhodes,Greece.

[Pieraccini et al., 1991] Pieraccini, R., Levin, E., and Lee, C.-H. (1991). Stochastic representation of conceptual structure in the atis task. In *Proc Speech and Natural Language Workshop*, pages 121–124, Pacific Grove, CA. DARPA.

[Pulman, 1997] Pulman, S. (1997). Conversational games, belief revision and bayesian networks. Technical report.

[Sadek et al., 1995] Sadek, D., Bretier, P., Cadoret, V., Cozannet, A., Dupont, P., Ferrieux, P., and Panaget, F. (1995). A cooperative spoken dialogue system based on a rational agent model: A first implementation of the ags application. In *Proc ESCA/ETR Workshop on Spoken Dialogue Systems*, pages 145–148, Vigsø, Denmark.

[Scheffler and Young, 1999] Scheffler, K. and Young, S. (1999). Simulation of human-machine dialogues. Technical Report CUED/F-INFENG/TR 355, Cambridge University Engineering Dept.

[Seneff et al., 1992] Seneff, S., Meng, H., and Zue, V. (1992). Language modelling for recognition and understanding using layered bigrams. In *Int Conf Spoken Language Processing*, pages 317–320, Banff, Canada.

[Smith and Hipp, 1994] Smith, R. and Hipp, D. (1994). *Spoken Natural Language Dialog Systems: A Practical Approach*. Oxford University Press.

[Sutton and Barto, 1998] Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass.

[Ward, 1991] Ward, W. (1991). Understanding spontaneous speech. In *Proc ICASSP 91*, pages 365–368, Toronto, Canada.

[Young, 1996] Young, S. (1996). Large vocabulary continuous speech recognition. *IEEE Signal Processing Magazine*, 13(5):45–57.

[Young and Proctor, 1989] Young, S. and Proctor, C. (1989). The design and implementation of dialogue control in voice operated database inquiry systems. *Computer Speech and Language*, 3(4):329–353.

[Young et al., 1991] Young, S., Russell, N., and Thornton, J. (1991). The use of syntax and multiple alternatives in the vodis voice operated database inquiry system. *Computer Speech and Language*, 5(1):65–80.

[Zue, 1997] Zue, V. (1997). Conversational interfaces: Advances and challenges. In *Proc Eurospeech*, pages 9–14, Rhodes, Greece.