The Statistical Approach to the Design of Spoken Dialogue Systems

Technical Report CUED/F-INFENG/TR.433

Steve Young (sjy@eng.cam.ac.uk)

Cambridge University Engineering Department
Trumpington Street, Cambridge, CB1 2PZ, England

September, 2002

1 Introduction

There is currently much interest in building interactive human-computer interfaces which involve spoken input and output. Some examples are computer-assisted learning systems, spoken access to information, voice-control of systems such as in in-car navigation and multi-modal systems such as might be used in the next generation of PDA.

Traditional approaches to building SDS have typically involved using a speech recogniser to transform speech to words and from then on treating the problem as essentially an exercise in symbolic programming. The system implementation will then depend heavily on the use of hand-crafted task grammars and flow-charted dialogue management strategies.

Although such systems can be effective, the "authoring cost" is nevertheless high and the resulting systems are often fragile. The lesson of speech recognition development over the last two decades has been that there is much to be gained by modelling processes statistically and estimating model parameters from real data. Because such systems explicitly model variance, they are naturally more robust and furthermore, they can also adapt automatically on-line.

The aim of this technical report is to explore the extent to which the paradigm used for speech recognition can be extended to cover the design and implementation of complete spoken dialogue systems.

The report begins in the next section with a brief overview of a statistical model of spoken dialogue. Subsequent sections then explore speech understanding, semantic decoding, dialogue management and response generation in more detail.

2 A Statistical Model of Spoken Dialogue

A statistical model of a spoken dialogue system is shown in Figure 1. The system operates cyclically. It begins with a default, system-initiated, dialogue act A_s which is converted to an acoustic signal $\boldsymbol{X}_o$ inviting the user to speak. Based on the current user state[1], S_u, the user generates a signal $\boldsymbol{X}_i$ which is corrupted by noise before being input to a speech understanding component as the acoustic signal $\boldsymbol{Y}_i$. The noisy speech signal $\boldsymbol{Y}_i$ is decoded to give a set of dialogue acts A_u. These dialogue acts are interpreted causing the system state S_s to be updated[2]. The system's next action depends on the belief state B. This belief state encodes the information in the user's state S_u and

[1]The user state represents the user's current beliefs, desires and intentions

[2]The system state encodes the system's beliefs, the plan being followed, the dialogue history, relevant environmental factors, etc.

the system state S_s needed to take the appropriate next action. There will usually be uncertainty in this estimation, and hence in the general case, B will not be a single discrete variable, rather it will be a probability distribution over the combined event spaces of S_s and S_u. Based on the system's new updated belief state B, a new system dialogue act A_s is generated and the cycle repeats.

Note that the dialogue management function consists of two key steps: firstly the current belief state B is estimated, and secondly, based on this belief state, another action A_s is taken. The mapping from B to A_s depends on the dialogue policy Π and in the statistical approach to dialogue management, optimisation of the policy Π is a key goal. From this perspective, Figure 1 should be viewed as follows. Optimisation of the dialogue policy is essentially trivial if the dialogue manager has perfect knowledge of both the user state S_u, and the system state S_s. In practice, the system state S_s is either directly accessible or it depends on S_u. Hence, the real problem is finding out S_u. To do this, the dialogue manager repeatedly 'prods' the user by asking questions A_s and observes the (noise corrupted) responses A_u. Based on these responses, the Dialogue Manager gradually reduces its uncertainty in S to the point where it is able to satisfy the user's desires.

As an example, an automated pizza ordering system might say: "Pizza World - how may I help you?" and the user might respond: "I would like a pepperoni pizza, please". In this case, the decoded dialogue acts A_u might result in the belief state B being updated to record the likelihood that the user would like to purchase one pepperoni pizza. If the system state indicates that the ambient noise is low and the recogniser confidence is high, then the distribution of B might be quite narrow reflecting near certainty in the estimate of the user's desires and intentions. In such a case, the Dialogue Manager might simply ask if the user wants anything else before moving to close the transaction. If on the other hand, the system state indicates that the ambient noise is high and the recogniser confidence is low, then the distribution of B might be quite broad reflecting uncertainty in the estimate of the user's desires and intentions. In this case, the Dialogue Manager is likely to generate a confirm action to check explicitly that the assumed belief is correct.

This statistical model of an SDS is characterised by a minimal dependence on explicit rules, and a heavy dependence on learning from data. The speech understanding part can be designed using the traditional pattern-matching paradigm of proposing a parametric model for the posterior distribution $P(A_u|\boldsymbol{Y})$, estimating the parameters of this model using training data and then decoding using some form of MAP rule. The response generation can be implemented by modelling $P(\boldsymbol{X}_o|A_s)$ and then sampling this distribution. Finally, the dialogue itself can be represented by a Markov Decision Process (MDP) and a *reward* assigned to each state-action pair. The optimal dialogue management policy is then found by maximising the expected total reward using reinforcement learning. If the belief state B is discretised, then a regular discrete state MDP ensues and the optimal policy can be learnt fairly straightforwardly using dynamic programming. If the belief state B is continuous (ie it is a true distribution over S_u and S_s) then learning is rather more difficult. Continuous MDPs of this type are often called Partially Observable MDPs (POMDPS). Exact POMDP learning algorithms are essentially intractable however various approximate algorithms exist which can provide the basis for building practical systems.

3 Speech Understanding

The goal of the speech understanding component is to convert each input speech waveform $\boldsymbol{Y} = \boldsymbol{Y}_i$ into a set of dialog acts $A_u = \{a_1, a_2, \ldots\}$ given the current system state S_s i.e. we seek

$$\begin{aligned} \hat{A}_u &= \arg\max_{A_u} P(A_u|\boldsymbol{Y}, S_s) \\ &= \arg\max_{A_u} P(\boldsymbol{Y}|A_u, S_s) P(A_u|S_s) \end{aligned} \quad (1)$$

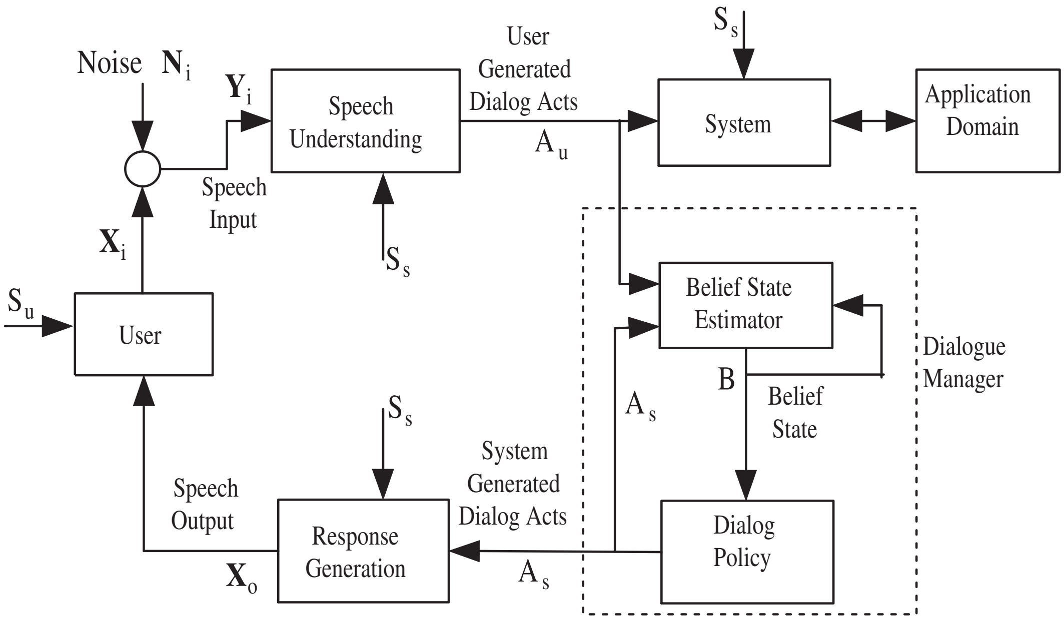

Figure 1: Block Diagram of a Spoken Dialogue System

Although it is not essential, it is convenient to identify the word sequence W carried by $\boldsymbol{Y}$

$$
\begin{aligned}
P(\boldsymbol{Y}|A_u, S_s) &= \sum_W P(\boldsymbol{Y}, W|A_u, S_s) \\
&= \sum_W P(\boldsymbol{Y}|W, A_u, S_s) P(W|A_u, S_s) && (2) \\
&\approx \arg\max_W P(\boldsymbol{Y}|W, A_u) P(W|A_u, S_s) && (3)
\end{aligned}
$$

where it is assumed that $\boldsymbol{Y}$ is conditionally independent of S_s if W is given. It is usual to ignore the dependence of $\boldsymbol{Y}$ on A_u for recognition purposes but take account of the dependence when computing confidences[3].

Substituting equation 3 into 1 (and ignoring the dependence of $\boldsymbol{Y}$ on A_u) gives

$$
\begin{aligned}
\hat{A}_u &= \arg\max_{A_u} \left\{ \arg\max_W \left\{ P(\boldsymbol{Y}|W) P(W|A_u, S_s) \right\} P(A_u|S_s) \right\} \\
&= \arg\max_{A_u} \left\{ \arg\max_W \left\{ P(\boldsymbol{Y}|W) P(W|S_s) P(A_u|W, S_s) \right\} \right\} && (4)
\end{aligned}
$$

Equation 4 is difficult to solve exactly, however, a suboptimal sequential solution can be achieved by first solving

$$
\hat{W} = \arg\max_W \left\{ P(\boldsymbol{Y}|W) P(W|S_s) \right\} \qquad (5)
$$

and then solving

$$
\hat{A}_u = \arg\max_{A_u} \left\{ P(A_u|\hat{W}, S_s) \right\} \qquad (6)
$$

[3]E.g. by using A_u to compute semantic next best scores.

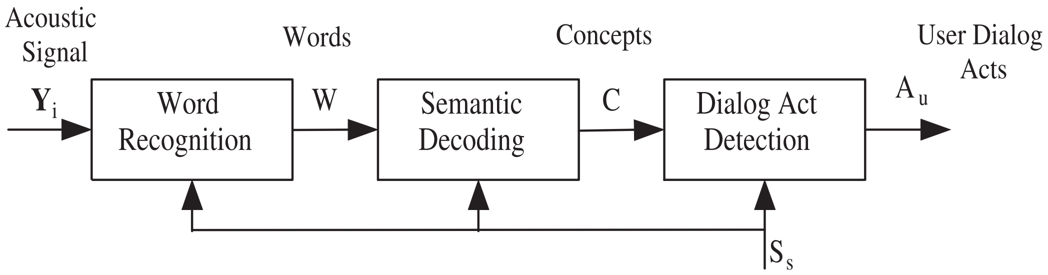

Figure 2: Speech Understanding

Equations 5 and 6 show that the semantic recognition problem can be factored into a two stage process of first recognising the underlying word string and then identifying the intended dialog acts from that string. The extent of the approximation incurred by this sequential decoding depends on how ambiguous the W are. In the exact case of equation 4, S_s will be used to disambiguate word string hypotheses when the acoustic and language models are not able to do so by themselves. The effect of the approximation can be reduced by retaining a word lattice in place of the single best string $\hat{W}$.

In both stages, the dependence on the current system state S_s is clearly shown. For the first speech recognition stage, this indicates that there are potential benefits from making the recogniser's language model dependent on the dialogue context. That is, if a set of specific dialog acts are expected then the language model should predict them. For the second semantic extraction stage, the system state is clearly crucial for handling ambiguity and identifying underspecified dialogue acts.

In order to evaluate equation 6, it is convenient to introduce another intermediate representation.

$$\begin{aligned} P(A_u|\hat{W}, S_s) &= \sum_C P(A_u, C|\hat{W}, S_s) \\ &= \sum_C P(A_u|C, S_s) P(C|\hat{W}, S_s) \end{aligned} \tag{7}$$

where C represents the set of semantic concepts encoded within the word sequence W. Thus, finally we can determine the required set of speech acts from

$$\hat{A}_u = \arg\max_{A_u} \left\{ \operatorname*{argmax}_C \left\{ P(A_u|C, S_s) P(C|\hat{W}, S_s) \right\} \right\} \tag{8}$$

which if we once again decode sequentially, simplifies to

$$\hat{C} = \operatorname*{argmax}_C \left\{ P(C|\hat{W}, S_s) \right\} \tag{9}$$

and then

$$\hat{A}_u = \arg\max_{A_u} \left\{ P(A_u|\hat{C}, S_s) \right\} \tag{10}$$

Equation 9 represents the process of semantic extraction which is commonly implemented using a hidden Markov model (section 4). Equation 10 represents the process of combining the system state with the decoded semantics to find the most likely dialog acts. This is commonly implemented using Bayesian Networks (section 5).

The combination of equations 5, 9 and 10 represent a complete decoding chain from acoustic signal to dialog act. It is illustrated in Figure 2. It represents a fully statistical approach to speech understanding and the sections below explore its constituents in more detail.

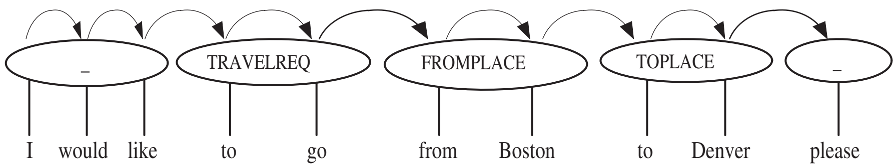

Figure 3: Discrete Markov Model Representation of Semantics

4 Semantic Decoding

This section explores the solution of equation 9 i.e. given the current system state S_s and the most likely word sequence $\hat{W}$ what is the most likely set of semantic concepts $\hat{C}$? Whereas the representation of a sentence W as a sequence of words is fairly uncontroversial, there are many ways of encoding the semantics C.

4.1 Discrete Markov Model

One of the simplest approaches to semantic decoding is to assume that each word w in an utterance W is associated with a single discrete concept c [1, 2]. For example, an utterance such as

User: I would like to go from Boston to Denver, please.

might be decoded as follows

```
TRAVELREQ(to go) FROMPLACE(from Boston) TOPLACE(to Denver)
```

where TRAVELREQ, FROMPLACE and TOPLACE are concepts and irrelevant words such as "I would like" and "please" have been discarded. This representation is illustrated in Fig 3 where the arrows denote the state transitions which occur as each word is processed and each semantic concept is associated with a discrete state. Irrelevant filler words are associated with a dummy concept (denoted here by _)which can be discarded after decoding. This will be referred to as the *flat concept* model and its decoder is a *finite state tagger (FST)*

Under this model of semantic decoding, equation 9 can be rewritten as follows where $W_{t_0}^{t_1}$ denotes the word sub-sequence $w_{t_0} \dots w_{t_1}$, and where $C_{t_0}^{t_1}$ denotes the concept sub-sequence $c_{t_0} \dots c_{t_1}$ [4]

$$\begin{aligned}
\hat{C} &= \operatorname*{argmax}_C \{P(C|W, S_s)\} && (11)\\
&= \operatorname*{argmax}_C \{P(W|C)P(C|S_s)\} && (12)\\
&= \operatorname*{argmax}_C \left\{ \prod_{t=1}^{T} P(w_t|W_1^{t-1}, C_1^t)P(c_t|C_1^{t-1}, S_s) \right\} && (13)\\
&\approx \operatorname*{argmax}_C \left\{ \prod_{t=1}^{T} P(w_t|w_{t-1} \dots w_{t-n+1}, c_t)P(c_t|c_{t-1} \dots c_{t-m+1}, S_s) \right\} && (14)
\end{aligned}$$

In equation 12, $P(C|S_s)$ is often referred to as the *semantic model* and $P(W|C)$ is referred to as the *lexical model*. The sequence of concepts $c_1 \dots c_T$ is modelled as an m-gram conditioned on the current belief state and the words are modelled as an n-gram conditioned on the semantic concepts c_t. If $m = 2$ and $n = 1$, the result is a conventional 1st order Markov model with states labelled by semantic concepts and transitions given by the concept bigram probabilities (see figure 4).

A discrete Markov model-based semantic decoder of this sort can be trained by providing labelled training data. If each training sentence is explicitly annotated as in

[4]In practice start and end sentence markers are used to enable prediction of the first word and last word of the sentence. This detail is omitted here for clarity.

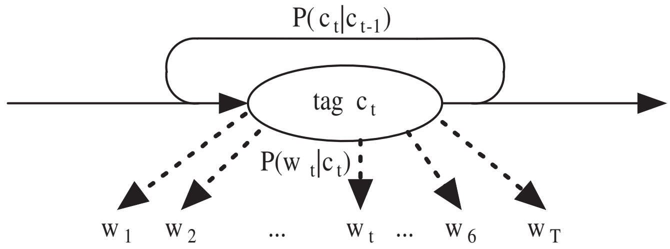

Figure 4: Structure of a 1st Order Markov Model for Semantic Decoding

```
_(I would like) TRAVELREQ(to go) FROMPLACE(from Boston)
TOPLACE(to Denver) _(please)
```

then the model parameters can be simply estimated by counting events and smoothing.

In data-base inquiry type systems where an SQL query must eventually be generated from query utterances, the high cost of annotating training data can be reduced by using a form of boot-strapping[3]. Instead of annotating the training data explicitly, the dialog system builder instead supplies the desired SQL query corresponding to each training utterance. This is the system development approach adopted by the ATIS programme [4, 5, 6]. In this technique, an initial model is trained using a small set of hand-annotated data. A larger set of utterances (without annotations but with corresponding SQL queries) is then automatically annotated using the initial model. Each newly annotated training utterance is then used to generate an SQL query and if it matches the supplied reference query, the utterance and its annotation are added to the training set. The whole process is iterated until no further additions can be made to the training set. Finally, the annotations of the remaining utterances can be hand-corrected or the utterances can be discarded. The main advantage of this form of training data annotation is that it is independent of the choice of semantic representation. This allows more flexibility in the system development process and it allows annotation to be performed by non-specialists.

Finally in this section, it should be noted that equation 12 is just the joint probability $P(W, C|S_s)$. The decomposition given by equation 13 represents a a *top-down left-right* generative model i.e. moving from left to right, first c_t is chosen based on the previous values $c_{t-1}\ldots$ and then a word w_t is chosen based on c_t. Equation 12 can also be decomposed to give a *bottom-up left-right* generative model as follows

$$\hat{C} = \underset{C}{\operatorname{argmax}} \{P(W, C|S_s)\} \tag{15}$$

$$= \underset{C}{\operatorname{argmax}} \left\{ \prod_{t=1}^{T} P(w_t|W_1^{t-1}, S_s) P(c_t|W_1^t C_1^{t-1}, S_s) \right\} \tag{16}$$

$$\approx \underset{C}{\operatorname{argmax}} \left\{ \prod_{t=1}^{T} P(w_t|w_{t-1}\ldots w_{t-n+1}, S_s) P(c_t|w_t, c_{t-1}\ldots c_{t-m+1}, S_s) \right\} \tag{17}$$

In this case, w_t is chosen first based on the previous words $w_{t-1}w_{t-2}\ldots$, then c_t is chosen based on w_t and the previous concepts $c_{t-1}c_{t-2}\ldots$.

4.2 Stochastic Context-free Grammar Model

The simple discrete Markov model-based semantic decoder described in the previous section has a number of drawbacks which stem from the fact that the flat left-right structure does not allow any hierarchical grouping of the concepts. This lack of hierarchical grouping weakens the predictive

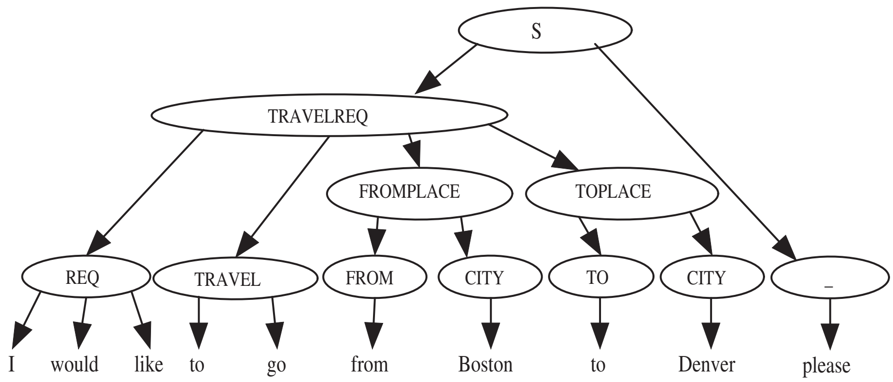

Figure 5: Stochastic Context-Free Grammar Representation of Semantics

power of the model since adjacent symbols are only weakly coupled and it reduces the expressive power of the representation since it cannot represent nested structures.

One approach to solving this problem is to convert the simple transition network shown in Fig 4 into a recursive transition network such that any concept-state can itself be a finite state network[7]. This extends the class of supported languages from regular to context-free and the model is therefore referred to as the *Stochastic Context-Free Grammar(SCFG)* model.

Fig 5 shows the form of semantic analysis supported by this model and it illustrates the two key advantages of the hierarchical SCFG model compared with the flat discrete Markov model discussed earlier. Firstly, the TOPLACE and FROMPLACE *attributes* are explicitly linked to the *topic* TRAVELREQ. In more complex queries this type of explicit binding can be important. For example, in the utterance "I want to go to Dulles not to Dallas next Saturday from Denver" TOPLACE appears twice and the first instance needs to be associated with the final FROMPLACE clause. This is not possible with the discrete Markov model where in this case, the sequencing of concepts would suggest erroneously binding TOPLACE(Dallas) with the FROMPLACE.

Secondly, the ability to create *preterminal* symbols with a distinct function from marking semantic roles makes it possible to avoid fragmenting the training data through arbitrary partitioning of attribute values. For example, in Fig 3, cities are divided between the TOPLACE and FROMPLACE concepts. The hierarchical analysis shown in Fig 5, however, allows city names to be naturally grouped into a single class via the preterminal CITY resulting in more robust statistical estimates.

Given a decoded semantic tree C in this hierarchical SCFG model, the sequence of preterminal nodes can be identified and the probabilities corresponding to the lexical model $P(W|C)$ computed as for the discrete Markov model. However, the probabilities for the semantic model $P(C|S_s)$ must be computed recursively and they are complex. Let $P(c,i,j)$ be the probability that concept c dominates the sequence of preterminal nodes c_i to c_j. If there are N preterminal nodes, $c_1 \ldots c_N$, then $P(C) = P(s,1,N)$ where s is the concept at the root of the tree. Also let $P(c \to c_1 \ldots c_Q)$ be the probability of the node for concept c directly generating the sequence $c_1 \ldots c_Q$. Then the so-called *inside probability* $P(c,i,j)$ is given recursively as

$$P(c,i,j) \quad = \quad \sum_{Q \leq j-i+1} \; \sum_{C_1^Q \in \{c*\}} \; \sum_{I_0^Q \in \{(i..j)^*\}} P(c \to c_1 \ldots c_Q) \prod_{q=1}^{Q} P(c, I(q-1), I(q)) \qquad (18)$$

where I_0^Q is intended to represent a set of Q integers which partition the sequence $c_i \ldots c_j$ into Q subsequences such that $I(0) = i$ and $I(Q) = j$.

For the case of binary branching, the above simplifies considerably to give

$$P(c,i,j) \quad = \quad \sum_{c_l,c_r} \sum_{t=i}^{j-1} P(c \rightarrow c_l c_r) P(c_l,i,t) P(c_r,t+1,j) \tag{19}$$

and this is now the inside probability of the *Inside-Outside algorithm* which when combined with the complementary *outside probability* provides the basis for parameter estimation using EM [8]. Recently, this same problem has been reformulated from the perspective of hierarchical HMMs to give further insights into the computational issues inherent in this class of recursive model [9].

In [7] the estimation problem is greatly simplified by using fully-annotated corpora. For understanding, a modified version of the Earley parsing algorithm is used [10] which generates the sequence of concepts C_1^Q or *parse path* corresponding to a depth-first scan of the entire parse tree. In this case, the probabilities of the semantic and lexical models are computed in an efficient interleaved manner as

$$P(C_1^Q) \quad = \quad \prod_{q=1}^{Q} \left\{ \begin{array}{ll} P(c_q|c_{q-1},c*) & \text{if } c* \text{ is in semantic model} \\ P(w_q|w_{q-1},c*) & \text{if } c* \text{ is in lexical model} \end{array} \right. \tag{20}$$

where $c*$ denotes the concept corresponding to the current model along the parse path. Note that with regard to equation 14, this model formulation sets $m = 2$ and $n = 2$, ie both the semantic and lexical models are context-dependent bigrams.

In common with all recursive models of this sort, SCFG-based models suffer from a variety of theoretical and practical problems. In particular, they are not left-right and their recursive nature makes them computationally intractable, especially for implementing EM-like parameter estimation. They also suffer from normalisation problems. For any given input word sequence, there will be many potential parses of differing derivation lengths. When total likelihoods are calculated over complete paths, this is not a problem since summations over all possible paths will always sum to 1. However, when Viterbi approximations are used on partial paths, this is no longer the case. The probabilities of utterances with complex parse trees consisting of many nodes are underestimated and competing partial paths unavoidably represent differing spans of the input.

4.3 Vector-state Markov Model

The key feature of a 1st order Markov model is that the current state at time t holds all of the information needed to account for the observation at time t and the transition to a new state at time $t+1$. It is this property which gives the model its mathematical simplicity.

Given the criticisms of the discrete Markov model made in section 4.2, one obvious way to improve the model as a basis of semantic representation is to extend the state space to record more context. The information in a parse tree relating to any single word can be stored as a vector of node names starting from the preterminal and ending at the root node. For example, in Fig 5, the word Boston is described by the semantic vector [City,FromPlace,TravelReq,S] and the complete parse tree can be represented by a sequence of such vectors. Indeed, provided that all node names in a parse tree are distinct there is a one-one mapping between semantic vector sequences and parse trees. Fig 6 shows the vector-state representation of the concept tree structure shown in Fig 5.

Given an upper limit on tree depth, any SCFG formalism can be converted into a 1st order Vector-state Markov model. The problem, however, is that in doing so the state space rapidly becomes huge. One way to alleviate this is to limit the possible transitions from one state to another thereby reducing the size of the semantic model (transition matrix), and indirectly the size of the lexical model (output distribution matrix).

Taking a lead from conventional parsing, the state vector can be viewed as a stack and each transition constrained to take the form of a stack shift followed by a push of one or more concepts

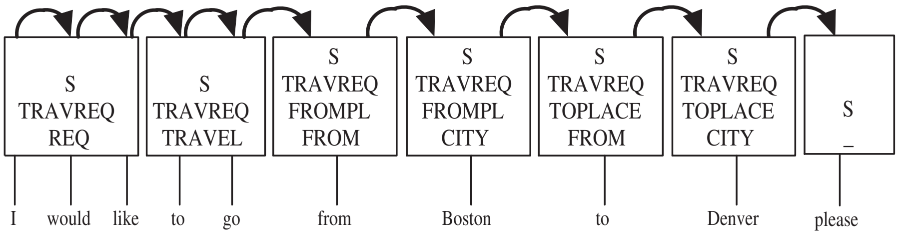

Figure 6: Vector-state Equivalent of Parse Tree in Fig 5

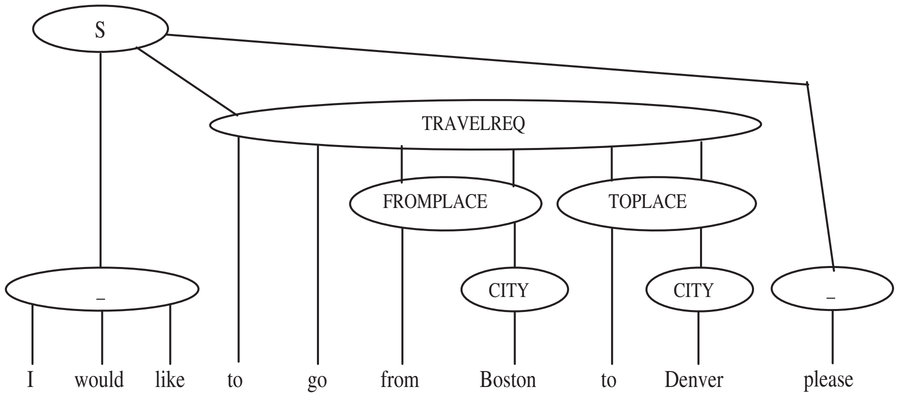

Figure 7: Example of a Right Branching Parse Tree

relating to the next word. A particularly simple form of parse arises if at most one new concept can be pushed per input word. In effect, the parser then behaves like a discrete Markov model extended to allow concept tags to be saved on the stack to provide a history mechanism. The effect of this on the *parse tree* of semantic concepts is to make it right-branching. As an example, Fig 7 shows a right-branching analysis of the utterance "I would like to go from Boston to Denver please". Notice that proceeding left to right, at most one new concept is introduced with each new word. Fig 8 shows the same parse tree represented as a vector sequence.

Given that word w_{t-1} is labelled by the concept vector $\boldsymbol{c}_{t-1}$, and the immediate parent for w_t is c_w, then the full concept vector $\boldsymbol{c}_t$ at time t is determined by the following sequence of actions:

1. $\boldsymbol{c}_{t-1}$ is shifted by n positions such that $c_t[i] = c_{t-1}[i+n]$ for all $i > 1$ such that $c_{t-1}[i+n]$ exists. n is limited to the range $-1, 0, 1, .., D_{t-1}$ where D_t is the stack size at time t.

2. $c_t[1]$ is set equal to c_w

In the above, setting $n = -1$ is equivalent to pushing the stack of concept nodes by one element, setting n to a positive value is equivalent to popping n concepts off the stack. Setting $n = 0$ leaves the stack the same size, but the immediate parent is replaced by c_w.

The generative process associated with this model is bounded and consists of three steps for each word position t: (a) choose a value for n_t; (b) select a new immediate parent $c_t[1]$; (c) select a word w_t. As with the discrete Markov model, the probability distribution $P(W, C, N)$ can be decomposed either *top-down* or *bottom-up*. The top-down decomposition is

$$P(W,C,N) = \prod_{t=1}^{T} P(n_t|W_1^{t-1}, C_1^{t-1}) P(c_t[1]|W_1^{t-1}, C_1^{t-1}, n_t) P(w_t|W_1^{t-1}, C_1^t) \quad (21)$$

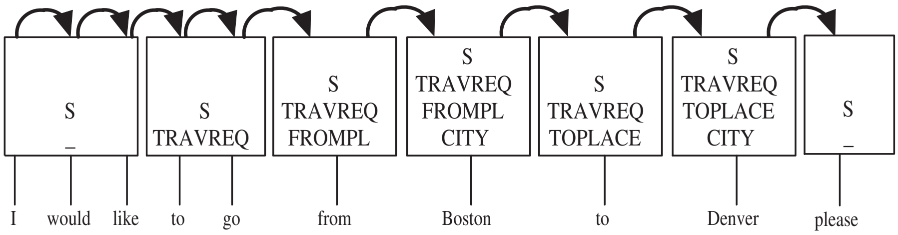

Figure 8: Vector-state Equivalent of Parse Tree in Fig 7

and this can be approximated by

$$
\begin{aligned}
P(n_t|W_1^{t-1}, C_1^{t-1}) &= P(n_t|\boldsymbol{c}_{t-1}) & (22) \\
P(c_t[1]|W_1^{t-1}, C_1^{t-1}, n_t) &= P(c_t[1]|c_t[2..D_t]) & (23) \\
P(w_t|W_1^{t-1}, C_1^t) &= P(w_t|\boldsymbol{c}_t) & (24)
\end{aligned}
$$

$c_t[2, \ldots, D_t]$ denotes stack elements $2, 3, .., D_t$.[5] Note that this slice of the stack at time t is deterministic given $\boldsymbol{c}_{t-1}$ and n_t.

Unlike the general hierarchical model, this probabilistic model is well-suited to left-right decoding. Since each partial path covering W_t contains exactly the same number of probabilities, paths can be compared directly without normalisation and pruning. Compared to the general SCFG model, the effective state-space and model size is much reduced. If there are on average $\mathcal{C}$ possible node labels (concepts) at any node and the maximum stack depth is D, then the transition matrix for the unconstrained model has $\mathcal{O}(\mathcal{C}^{2D})$ parameters. The effective transition matrix for the constrained right branching case is represented by equations 22 and 23 which require approximately $\mathcal{O}((D+3)\mathcal{C}^D)$ parameters which, though still large, is a significantly smaller number.

4.4 The Hidden Vector-state Markov Model

In the preceding sections, it has been assumed that the training corpus is fully annotated and parameter estimation is simply a matter of event counting and smoothing. In practice, however, the provision of fully annotated data can be prohibitively expensive to produce. The bootstrapping approach described in section 4.1 can go some way to alleviate this but the need for a sizeable quantity of fully annotated data remains. Methods of training using unannotated or partially annotated data are therefore of considerable interest. Not only can they reduce cost, but they can also remove the need to commit to an explicit representation for the parse of each utterance.

Following the practice established for training hidden Markov models, the general approach to training using partially annotated data is to treat the model state as a hidden variable, and then use Expectation-Maximisation (EM) to estimate the model parameters from the observed data. Although in principle this approach can be applied to completely unannotated data, in practice this is not a realistic option. Empirical studies in very restricted domains have shown that the phrase structures recovered automatically by EM-based algorithms such as the Inside-Outside algorithm, bear little resemblance to any linguistically intuitive structure[11]. Thus, it appears that constraints in the form of *a priori* knowledge and/or partial annotations must be used.

In the context of spoken dialogue systems, it is natural to ask what sorts of *a priori* knowledge or partial annotations can a system designer reasonably provide. In this respect there are two obvious possibilities:

- provision of a set of domain specific lexical classes. For example, in a flight information system, typical classes might be CITY={Boston, Denver, New York, ... }, AIRPORTS={Dulles,

[5] Stack element $c_t[D_t]$ is always the root node S.

La Guardia, Logan, ...} etc. These domain specific classes are usually extracted automatically from the domain database schema. They can also be augmented by generic classes covering times, dates, etc. Given these classes, all word sequences W are preprocessed and all detected class members replaced by their class names. This simultaneously reduces the effective vocabulary size and introduces constraints on the plausible analyses of W.

- an abstract semantics for each utterance. This would provide a list of applicable concepts and the dominance relationships between concepts. However, it would take no account of word order or attempt to suggest an interpretation for all parts of the utterance. For example, all of the following sentences would have the same abstract semantics:

1. I want to go to Chicago to arrive around 11am.
2. I need to arrive about noon in New York.
3. I have to be in Boston at 10am.
4. Find flights arriving in Dallas mid-morning.

$\Bigg\}$

```
TRAVELREQ(
  TOPLACE(
    CITY,
    TIMESPEC(TIME)
))
```

The way that such abstract semantics can be used in training is discussed further below.

Letting the complete set of model parameters be denoted by λ, EM-based parameter estimation requires the expression

$$\sum_{C,N} P(C,N|W,\lambda)\log P(W,C,N|\hat{\lambda}) \tag{25}$$

to be maximised wrt to $\hat{\lambda}$. The approximation for $P(W,C,N)$ given by equations 22 to 24 is

$$P(W,C,N) = \prod_t P(n_t|\boldsymbol{c}_{t-1})P(c_t[1]|c_t[2..D_t])P(w_t|\boldsymbol{c}_t) \tag{26}$$

Plugging this expression into 25 and differentiating leads to the following reestimation formulae:

$$\hat{P}(n|\boldsymbol{c}') = \frac{\sum_t P(n_t=n,\boldsymbol{c}_{t-1}=\boldsymbol{c}'|W,\lambda)}{\sum_t P(\boldsymbol{c}_{t-1}=\boldsymbol{c}'|W,\lambda)} \tag{27}$$

$$\hat{P}(c[1]|c[2..D]) = \frac{\sum_t P(\boldsymbol{c}_t=\boldsymbol{c}|W,\lambda)}{\sum_t P(c_t[2..D]=c[2..D]|W,\lambda)} \tag{28}$$

$$\hat{P}(w|\boldsymbol{c}) = \frac{\sum_t P(\boldsymbol{c}_t=\boldsymbol{c}|W,\lambda)\delta(w=w_t)}{\sum_t P(\boldsymbol{c}_t=\boldsymbol{c}|W,\lambda)} \tag{29}$$

where $\delta(w=w_t)$ is one iff the word at time t is w, otherwise it is zero.

The key components of the above reestimation formulae are the likelihoods $P(c_t[2..D]=c[2..D]|W,\lambda)$, $P(n_t=n,\boldsymbol{c}_{t-1}=\boldsymbol{c}'|W,\lambda)$ and $P(\boldsymbol{c}_t=\boldsymbol{c}|W,\lambda)$. These can be efficiently calculated using the forward-backward algorithm. First define, the forward and backward probabilities, α and β (where the model parameters λ are assumed given)

$$\alpha_{\boldsymbol{c}}(t) = P(W_1^t,\boldsymbol{c}_t=\boldsymbol{c}) \tag{30}$$

$$\beta_{\boldsymbol{c}}(t) = P(W_{t+1}^T|\boldsymbol{c}_t=\boldsymbol{c}) \tag{31}$$

Then, omitting the conditioning on W and λ for clarity, the required likelihoods are

$$P(c_t[2..D]=c[2..D]) = \sum_{\{\boldsymbol{c}|c_t[2..D]=c[2..D]\}} \alpha_{\boldsymbol{c}}(t)\beta_{\boldsymbol{c}}(t) \tag{32}$$

$$P(n_t=n,\boldsymbol{c}_{t-1}=\boldsymbol{c}') = \alpha_{\boldsymbol{c}'}(t-1)P(n_t=n|\boldsymbol{c}_{t-1}=\boldsymbol{c}')\sum_{\{\boldsymbol{c}|\boldsymbol{c}'\overset{*}{\Rightarrow}\boldsymbol{c}\}} P(c[1]|c[2..D])P(w_t|\boldsymbol{c})\beta_{\boldsymbol{c}}(t) \tag{33}$$

$$P(\boldsymbol{c}_t=\boldsymbol{c}) = \alpha_{\boldsymbol{c}}(t)\beta_{\boldsymbol{c}}(t) \tag{34}$$

where

$$P(\boldsymbol{c}|\boldsymbol{c}') \quad = \quad P(n^*|\boldsymbol{c}')P(c[1]|c[2..D]) \tag{35}$$

and where n^* is the value of stack shift n necessary to map $\boldsymbol{c}'$ into $\boldsymbol{c}$.

Finally, the forward and backward probabilities are defined recursively as follows

$$\begin{aligned} \alpha_{\boldsymbol{c}}(t) &= \{ \sum_{\{\boldsymbol{c}'|\boldsymbol{c}'\overset{*}{\Rightarrow}\boldsymbol{c}\}} \alpha_{\boldsymbol{c}'}(t-1)P(\boldsymbol{c}|\boldsymbol{c}')\}P(w_t|\boldsymbol{c}) && (36) \\ \beta_{\boldsymbol{c}}(t) &= \sum_{\{\boldsymbol{c}''|\boldsymbol{c}\overset{*}{\Rightarrow}\boldsymbol{c}''\}} P(\boldsymbol{c}''|\boldsymbol{c})P(w_{t+1}|\boldsymbol{c}'')\beta_{\boldsymbol{c}''}(t+1) && (37) \end{aligned}$$

In all of the above, a single training utterance is assumed. The extension to the more realistic case of multiple utterances follows trivially from the equivalent case in standard HMMs.

Returning now to the issue of using partially annotated training data, the notation $\boldsymbol{c}' \overset{*}{\Rightarrow} \boldsymbol{c}$ is intended to denote "all state vectors $\boldsymbol{c}'$ from which the state vector $\boldsymbol{c}$ can legally be derived". This therefore is a possible place to apply constraints. For example, a derivation of the form

$$[\ldots, A, \ldots, S] \Rightarrow [B, \ldots, A, \ldots, S] \tag{38}$$

should only be allowed if the corresponding abstract semantic description for that utterance includes a node B dominated by a node A. To apply, strict constraints of this form, some provision must be made to account for irrelevant input, hence, some form of "dummy" concept should be allowed everywhere.

An alternative approach to applying hard constraints is to construct an *a priori* annotation model λ^r for each training utterance r which allows only the transitions prescribed by the abstract semantic annotation. Each iteration of EM then uses an interpolated set of model parameters to compute the forward-backward probabilities and likelihoods

$$\lambda_t = \gamma_t \lambda_{t-1} + (1 - \gamma_t)\lambda^r \tag{39}$$

where *gamma* controls the learning rate. It is initially small putting most weight on the *a priori* annotation model, but it increases towards 1 as learning progresses.

4.5 Model Adaptation

The previous sections have briefly described various approaches to parameter estimation using unannotated training data. Regardless of how much training data is available, a semantic decoder will perform badly if the training data does not well-represent the test data. Where a mis-match between training and test is inevitable, then unsupervised adaptation can be used to learn a transformation of the models which maximises the likelihood of the training data. This is analagous to methods such as MLLR applied to acoustic models[12].

An example of unsupervised model adaptation is given in [13] where the general idea of MLLR is carried over to the discrete state. Suppose a model for $P(W, C)$ exists with J component distributions $\{P_j\}$ each of dimension K, then given some adaptation data $\{W_l\}$, a Markov matrix Q can be found to transform the model $\hat{P}_j = P_j Q$ such that the log likelihood

$$LL(Q) \quad = \quad \sum_{j=1}^{J} \sum_{k=1}^{K} \sigma_j(k) \log \hat{P}_j(k) \tag{40}$$

is maximised. In this equation, $\sigma_j(k)$ is the total count of the events associated with the k^{th} component of P_j summed across the decoding of all adaptation utterances W_l. Since Q is constrained to be Markov ie

$$\sum_{k=1}^{K} q_{ik} = 1 \quad \forall i \tag{41}$$

$$q_{ik} \geq 0 \quad \forall i,j \tag{42}$$

the $\hat{P}_j$ are guaranteed to remain valid probability distributions. It also follows from the Markov property that all possible values of Q form a convex set.

As an example, this technique can be applied to the Hidden Vector State (HVS) model by estimating a separate global Q matrix for each of the three component distributions[6] i.e. from equation 26

$$LL = \sum_l \sum_t \left\{ \hat{P}_n(n_t|\boldsymbol{c}_{t-1}) + \hat{P}_c(c_t[1]|c_t[2..D_t])) + \hat{P}_w(w_t|\boldsymbol{c}_t) \right\} \tag{43}$$

$$= LL_n + LL_c + LL_w \tag{44}$$

and then each term can be optimised independently. For example,

$$LL_n = \sum_j \sum_{k=-1}^{D} \sigma_j^{(k)} P_n(k|\boldsymbol{c}_j) q_{kj} \tag{45}$$

where j ranges over all possible values of concept vector $\boldsymbol{c}_j$, k ranges over all possible values of stack pop and $\sigma_j^{(k)}$ is the number of times that a stack pop of k occurred in the adaptation data when the previous concept vector was $\boldsymbol{c}_j$. Since each row of $Q^{(n)}$ sums to one, equation 45 can be straightforwardly optimised row by row.

5 Dialog Acts and Meaning Representation

For the class of limited domain systems considered here, meaning can be adequately represented by simple semantic tree structures. For example, Figure 9(a) shows a tree structure representing the partially instantiated user query for a pizza. This might represent the state immediately following the user saying, "I would like two Ham and Cheese pizza's please", and it might be followed by the system saying, "You want two Ham and Cheese pizza's. What size?". Here, the system chooses to use implicit confirmation because the confidence levels (shown in brackets) are neither very high nor very low. If the user simply responds "Large." then she confirms the uncertain information and provides the final piece of information - the size. Note that this single word response actually encodes two dialogue acts: firstly, silence implies confirmation, and "Large" gives a value for size.

More formally, each user utterance Y can be decoded as one or more dialog acts $A_u = \{a_1, a_2, ...\}$. Each dialog act a_j consists of a functor of the form $a_j(c_1 = v_1, c_2 = v_2, \ldots)$ where the c_i are the semantic concepts discussed in the last section and the v_i are the corresponding values. For example, the utterance "I would like two Ham and Cheese pizza's please" would be decoded as

```
pizza_order {
     topping = Ham&Cheese
     quantity = 2
}
```

Concept names can be dotted to represent the tree hierarchy. For example, referring to figure 9(b), the user utterance "I want to go to London tomorrow" might be decoded as

[6]It is straightforward to add multiple transforms if there is sufficient adaptation data to reliably estimate their parameters

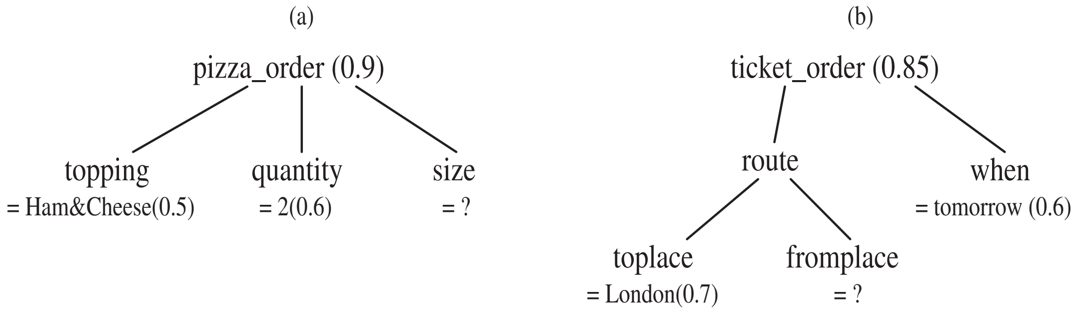

Figure 9: Example Fragment of a Semantic Tree

```
ticket_order {
    route.toplace = London
    when = tomorrow
}
```

In general, identification of the act a_j involves solving equation 10. In some cases, the semantic decoder might identify the act explicitly, as when for example, the user says, "I want to buy a ticket to go to London tomorrow" and this is then decoded as

```
ticket_order {
    goal = ticket_order
    route.toplace = London
    when = tomorrow
}
```

Otherwise the intended dialogue act must be inferred from the provided semantic concepts. For cases where semantic decoding involves dialogue act specific rules, then the appropriate act can be inferred from the set of rules which provide the best match. A similar approach is taken in BBN's HUM system where a surface meaning representation M_s is first determined based on the N-best semantic parse trees T, and then the most likely dialog act is selected based on M_s and the history H[14].

If A_u is limited to being a single dialogue act, then binary decision trees can be used for act detection. In this case, the leaf nodes correspond to dialogue acts and each tree node q is labelled with a subset $C_q \subset \mathcal{C}$ implying the question "Is any input concept $c \in C_q$". Such a tree can easily be trained automatically from examples of actually occurring concept sets and the corresponding dialogue acts.

An alternative which provides a soft-classification and which allows detection of multiple dialogue acts uses Bayesian Networks (BNs) [15, 16]. In the approach developed by Meng and colleagues, one BN is defined for each possible dialogue act as shown in Fig.10. The conditional probabilities $P(c_k|a_j)$ and the priors $P(a_j)$ are learnt from training data. Given a set of input concepts (the evidence), then $P(a_j|c_1,..,c_M)$ is computed for each act a_j, and the most likely are selected as the decoded output. As shown by the dotted line in Fig.10, concepts are not all conditionally independent and performance can be improved by including *between-concept* dependencies. In [17], an automatic method is proposed for learning the most significant dependencies based on minimum description length. Once the evidence has been entered into the network, the posterior concept probabilities can be computed. A concept which has a high posterior but which was not supplied in the evidence is probably needed information, and should be asked for. Similarly, a concept with low probability but which was in the evidence might be spurious, and it should be confirmed with the user. Thus, BNs used in this way can assist with mixed initiative dialogue management[18, 19].

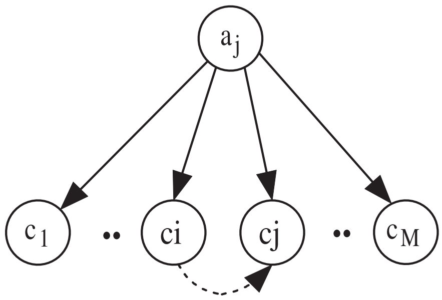

Figure 10: Dialogue Act Detection using a Bayesian Network

Finally, in this context the work of Bellegarda and Silverman based on latent semantic analysis should be mentioned[20]. They construct a matrix $W = \{w_{jk}\}$ which records the number of times that concept c_k occurred in an utterance conveying dialogue act a_j. They compute the SVD $W = USV^T$ and then at run-time they use U^T to map input concept vectors into a much reduced subspace where a simple distance metric can be used to identify likely dialogue acts. In fact they include words rather than concepts in the input vectors and dispense with semantic decoding altogether giving an effective and robust technique for simple domains. The difficulty with this approach, however, is that it does not readily scale to the case where a single utterance simultaneously encodes several dialogue acts.

6 Dialogue Management

6.1 States and Belief States

Referring back to Figure 1, the goal of the dialogue manager is to respond to incoming user dialogue acts A_u by updating its belief state B and then generating an appropriate response to the user in the form of a set of output system dialogue acts A_s. In the context of this section, these latter system dialogue acts will also be referred to as *system actions*.

In the general case, the system belief state B is a probability mass function over possible dialogue states S where a *dialogue state* encodes the full state of the entire dialogue, i.e. $S = S_u \otimes S_s$. Note that neither the true states of S_u nor S_s are known with certainty. S_u encodes the beliefs, desires and intentions of the user and S_s encodes the current assumptions of the system. The belief state explictly encodes this uncertainty in the form of a distribution, i.e.

$$B(S) = P(S|S \in \mathcal{S}) \quad \text{and} \quad \sum_{S \in \mathcal{S}} B(S) = 1 \tag{46}$$

where $\mathcal{S}$ is the set of all possible dialogue states.

The underlying dialogue state S must encode all that is relevant about the state of the dialogue. S is typically represented as a combination of lower cardinality *sub-state* variables, i.e.

$$S = s_1 \otimes s_2 \otimes s_3 \ldots \otimes s_N \tag{47}$$

where $|S| = \prod_{i=1}^{N} |s_i|$.

For example, the pizza ordering system mentioned in section 2 might have a state space factored into 6 sub-state variables:

substate	values	card.	description
s_1	cheese, ham, ...	#toppings	user's desired topping
s_2	small, medium, large	3	user's desired size
s_3	1,2,3,...	max qty	user's desired qty
s_4	cheese, ham, ...	#toppings	system's understood topping
s_5	small, medium, large	3	system's understood size
s_6	1,2,3,...	max qty	system's understood qty

Assuming 10 different toppings, and a maximum order of 6 pizzas, the cardinality of this state space is $6^2 * 3^2 * 4^2 = 32000$. If other state history and environmental information is added, then the cardinality rapidly becomes intractibly large.

In practice, the above representation of the system state as a mirror of the user's state is not very useful. The dialogue manager rarely needs to know the actual values of *form variables* such as topping, size, etc. What is more important is the status of these variables as viewed by the system i.e. are they known, and have they been grounded. Thus, a more realistic representation would be

substate	values	card.	description
s_1	cheese, ham, ...	#toppings	user's desired topping
s_2	small, medium, large	3	user's desired size
s_3	1,2,3,...	max qty	user's desired qty
s_4	unknown, known, grounded	3	system's status of topping variable
s_5	unknown, known, grounded	3	system's status of size variable
s_6	unknown, known, grounded	3	system's status of qty variable

The cardinality of S_u is of course unchanged by this, but the cardinality of the system component S_s is much reduced (to $3 * 3 * 3 = 27$). This is actually more helpful than it might at first appear since for many applications, the user state S_u is assumed to be constant and only the system state S_s actually changes during a dialogue (see section 6.3).

6.2 Markov Decision Processes

As noted in the overview in section 2, a complete dialogue system can be modelled as a Markov Decision Process (MDP) in which each dialogue exchange results in a state transition from S to S'[7].

Early work on MDPs in dialogue assumed that the complete state S was observable and that the state transition matrix $P(S'|A_s, S)$ could be estimated directly from data [21, 22]. Referring back to figure 1, this corresponds to the case where the belief state is discrete and some deterministic function of S. The system actions, A_s, are then directly dependent on S via a policy π

$$A_s = \pi(S) \tag{48}$$

The goal of modelling dialogue as an MDP is to be able to automatically optimise this dialogue policy π. To do this, an expected reward $r = R[S, A_s]$ is associated with each state and system action. The total expected reward at time t is then

$$R_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau \tag{49}$$

where $\gamma < 1$ is a discount factor to ensure that the expected reward remains finite. Rewards are generally chosen to reflect the characteristics required of the dialogue. For example, every action from a non-terminal state might have a small negative reward, achieving overall success

[7] In this section, primes are used as a shorthand to indicate updates at time $t+1$ relative to the current time t

might generate a large positive reward and entering a failure state might generate a large negative reward. Such a reward structure will attempt to find a successful outcome in the minimum number of steps.

If the combined user and system state are considered to be fully observable, then solution of the MDP problem is a straightforward application of classical reinforcement learning (see [23] for a good introduction). All solution methods depend on finding the value function

$$V^\pi(S) = E_\pi \{R_t | S_t = S\} \tag{50}$$

For any state S, this function gives the expected value of reward if the dialogue proceeds from this state to a terminal state using policy π[8]. Even more useful is the closely related function

$$Q^\pi(S, A) = E_\pi \{R_t | S_t = S, A_t = A)\} \tag{51}$$

which gives the expected value of reward if action A is taken from state S. If this Q function is known, then the policy π' determined by

$$\pi'(S) = \underset{A}{\operatorname{argmax}}\, Q^\pi(S, A) \tag{52}$$

is guaranteed to be better or equal to π. This provides a basis for policy optimisation. If the system transition function is known, then Q can be found by dynamic programming using an update rule of the form

$$Q(S, A) \leftarrow r(S, A) + \sum_{S'} P(S'|S, A)\gamma \max_{A'} Q(S', A') \tag{53}$$

If the transition function is not known, then methods based on sampling actual dialogues can be used. In particular, *temporal difference learning* is a technique in which the Q function is updated after every dialogue turn $(S, A) \rightarrow (S', A')$ by comparing the actual one-step reward with the reward predicted by the Q function

$$Q(S, A) \leftarrow Q(S, A) + \alpha[r(S, A) + \gamma Q(S', A') - Q(S, A)] \tag{54}$$

where α determines the learning rate. In order to ensure that Q is considered over all reasonable combinations of (S, A), it is necessary to allow the dialogue to deviate from the optimal policy occasionally. This is commonly done by adopting a stochastic ϵ-soft policy. For each pair (S, A), let $A^* = \operatorname{argmax}_A \ \{Q(S, A)\}$ then

$$\pi(S, A) = \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(S)| & \text{if } A = A^* \\ \epsilon/|\mathcal{A}(S)| & \text{otherwise} \end{cases} \tag{55}$$

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 explore with probability $\epsilon/|\mathcal{A}(S)|$ a non-optimal action. As $\epsilon \rightarrow 0$, then the soft policy hardens to the optimal deterministic policy [24].

This assumption of a fully observable state space is somewhat unrealistic since in practice the user does not always respond entirely predictably and the speech understanding component will make errors. The effect of this uncertainty can be compensated to some degree by introducing discrete confidence levels explictly into the state space. For example, every variable can have an associated confidence factor of low, medium or high. Provided these confidence factors can be determined accurately, then the dialogue strategy can learn to use them to modify its behaviour. Indeed, this has been the focus of much of the existing work on using MDP's to model dialogue systems. For example, Walker *et al* have modelled several real systems using simple MDPs, made

[8]Of course, dialogues have a fixed but *a priori* unknown number of steps. A simple way of dealing with this is to allow all terminal states to self-loop with zero reward. The total rewards can then be safely "summed to infinity" without concern for the actual dialogue length

tractable by reducing the state space to focus on characteristics of specific interest such as the prompt type and the confirmation type [25, 26]. They used training data collected from live use of the SDS running a random policy designed to explore the state-action space. They were able to obtain significant improvements in reward measure and they were also able to improve ASR performance by discovering more finely tuned strategies for interpreting confidence measures.

Gathering training data using traditional Wizard-of-Oz techniques or by implementing a prototype system implies fixing the state space in advance. Furthermore, data gathered this way is generally expensive since it requires real users. One way to avoid this problem is to simulate the user by adopting a two stage approach[27, 28]. Firstly, a user simulation model is trained using a small amount of data obtained from real users interacting with a prototype dialogue system. Policy optimisation can then be performed using synthetic data generated by the user model. In a similar spirit, the speech recognition and understanding process can also be simulated and then used as the basis for synthesising realistic state transition functions[29].

Even with the benefit of simulations for efficiently generating realisting training data, the state space must still be cut-down significantly if optimisation is to be tractable. The effect of this is inevitably to discard history information and thereby make the process non-Markovian. It has been found that using elligibility traces can compensate somewhat for this effect [30]. Nevertheless, directly observed discrete MDPs will always be a rather limited compromise solution to the problem of dialogue modelling. This is especially true in real systems where recognition error rates can be high and the need to explicitly model the system's uncertainty in the user's beliefs, desires and intentions cannot be ignored.

6.3 Partially Observable MDPs

The principle problem with all of the methods described in the last section is that the complete dialogue state is never in practice observable. More accurate modelling demands that at minimum the hidden nature of the user state S_u is acknowledged. Furthermore, due to recognition errors, the system state S_s is also uncertain. To handle this in a principled way, an architure similar to that shown in Figure 1 is required in which the dialogue is modelled by a partially observable Markov decision process (POMDP).

In a POMDP, system actions are taken on the basis of the belief state B described in section 6.1 rather than the dialogue state S. In the POMDP framework, the user dialogue acts are regarded as observations arising from a noise measurement process, where the true underlying signal is the user output X_i[9]. A basic assumption is that if the system had access to this X_i then it could implement an optimal deterministic policy via a simple rule-based planning program.

The belief state is linked to the dialogue state via the state transition matrix and an observation model thus

$$\begin{aligned}
B'(S') &= P(S'|A'_u, A_s, B) && (56)\\
&= P(A'_u|S', A_s, B)P(S'|A_s, B)/P(A'_u|A_s, B) && (57)\\
&= \underbrace{P(A'_u|S', A_s)}_{\substack{\text{Observation}\\ \text{Model}}} \sum_{S\in\mathcal{S}} \underbrace{P(S'|A_s, S)}_{\substack{\text{Transition}\\ \text{Model}}} B(S)/P(A'_u|A_s, B) && (58)
\end{aligned}$$

where the denominator is a normalising term given by

$$P(A'_u|A_s, B) \quad = \quad \sum_{S'\in\mathcal{S}} P(A'_u|S', A_s) \sum_{S\in\mathcal{S}} P(S'|A_s, S)P(S|B) \qquad (59)$$

[9] X_i is actually an acoustic waveform and would normally be shown in bold. Here, however, it is interpreted as a discrete variable representing the information encoded in $\boldsymbol{X}_i$

Thus, the belief state is updated at every dialogue cycle by transiting from S to S' weighted by the probability of being in state S, $B(S)$, and weighting the result by the probability of observing the decoded dialogue act A_u.

The reward function for POMDP's is similarly computed by taking the expectation of $r = R[S, A_s]$ with respect to $B(s)$ i.e. $r = \sum_S B(S)R[S, A_s]$ and the total expected reward remains as specified by equation 49.

As with regular discrete MDPs, the computation of the value function

$$V^{\pi}(B) = \mathcal{E}\left\{\sum_{t=0}^{\infty} \gamma^t r_t\right\} \tag{60}$$

for any given belief state B and policy π is the key to policy optimisation. The optimal policy $\pi*$ is stationary [31] and hence the optimal value function V^* can be represented by

$$V^*(B) \quad = \quad \max_{A_s}\left\{\sum_S B(S)R[S, A_s] + \gamma \sum_{A'_u} P(A'_u|B, A_s)V^*(B')\right\} \tag{61}$$

where the B' denotes the belief state that results from taking action A_s and observing the user response A'_u from original belief state B. Note that this equation is intuitively reasonable. The first term in brackets is the expected value of immediate reward wrt to the conditional distribution of $P(S)$ and the second term is the expected value of future rewards wrt to the conditional distribution $P(A'_u)$.

In the MDP case where the state distributions are discrete, the analogous equation to 61 could be solved iteratively since it is essentially a dynamic programming problem in discrete space (cf equation 53). Unfortunately, the solution of equation 61 by iterative re-estimation is far from straightforward since V is now a function of a continuous rather than discrete variable. Worse still, B is continuous in a very high dimensional space.

Exact solutions to equation 61 are computationally highly complex. Fortunately, however, the whole of B space does not need to be searched. As first pointed out by Sondik [32], since B is a vector of probability values, the value function can be represented in the form

$$V(B) \quad = \quad \max_{\nu \in \Gamma} B \cdot \nu \tag{62}$$

where Γ is a set of vectors the same dimension as B (i.e. $|S|$), defining hyperplanes in value $\times$ belief space. Each ν_i in Γ can be shown to represent a fixed policy and the max of these functions forms a convex piecewise-linear manifold. The implication of this is that, in principle, only a very few of the possible ν_i ever need to be searched. The trick is finding out which ones and there are many approaches to this[33, 34, 35, 36]. Unfortunately, these all become compuatationally intractable for realistic state spaces. More recently, approximate solutions have started to emerge which give useful results with significantly reduced computation[37, 38]. However, it is still too early to say what methods will best suit dialogue modelling applications.

The tractable computation of value functions is not the only problem inherent in the POMDP framework. The above also glosses over the difficulty of accurately estimating the observation model and the state transition models given limited training data. In order to deal with this, it is helpful to decompose these two models into subcomponents.

Firstly, the observation model can be decomposed by splitting the dialogue state S into the user and state subcomponents and introducing the actual user output X_i as a hidden variable[10]:

$$\begin{aligned} P(A_u|S', A_s) &= P(A'_u|A_s, S'_u, S_s) && (63) \\ &= \sum_{X'_i} \underbrace{P(A'_u|X'_i, S_s)}_{\text{Recognition Model}} \underbrace{P(X'_i|S'_u, A_s)}_{\text{User Output Model}} && (64) \end{aligned}$$

[10]This decomposition was suggested by Jason Williams

Note here the observation arises after the user state has been updated but before the system state has been updated. The observation model is now rather more tractable since the user output model can be inferred from relatively limited data and the recognition model is as given in section 3.

The state transition model is decomposed similarly

$$
\begin{aligned}
P(S'|A_s, S) &= P(S'_u, S'_s|A_s, S) && (65)\\
&= P(S'_u|A_s, S)P(S'_s|S'_u, A_s, S) && (66)\\
&= \underbrace{P(S'_u|A_s, S_u)}_{\text{User Transition Model}} \sum_{X'_i} \underbrace{P(S'_s|X'_i, A_s, S_s)}_{\text{Domain Model}} \underbrace{P(X'_i|S'_u, A_s)}_{\text{User Output Model}} && (67)
\end{aligned}
$$

Equation 66 decomposes transitions into a two-step process. Firstly, the user state is updated in response to the system dialogue act A_s and then the system state is updated in response to the updated user state and the system action A_s. Equation 67 further decomposes state transitions by again introducing the actual user output X_i as a hidden variable. In this case, the domain model updates the system state based on its assumed knowledge of the user output. This represents the deterministic planning agent which is assumed to be an integral part of any practical dialogue management software. It will have a value of 1 iff the updated system state S' is as predicted by the planner and zero otherwise.

As noted earlier, for many applications it is sufficient to assume that the user state S_u is fixed for the entire dialogue e.g. the user has a single fixed goal and does not change her beliefs, desires or intentions significantly during the course of the dialogue. In this case, the user transition model in equation 67 is unity. The belief state update equations can also be simplified in this case

$$
\begin{aligned}
B'(S') &= B'_u(S'_u) \otimes B'_s(S'_s) && (68)\\
&= P(S'_u, |A_s, A'_u, B)P(S'_s, |S'_u, A_s, A'_u, B) && (69)\\
&= \left\{\frac{P(A'_u|S'_u, A_s, B)}{P(A'_u|A_s, B)}\right\} P(S'_u|A_s, B)P(S'_s, |S'_u, A_s, A'_u, B) && (70)\\
&= \left\{\frac{P(A'_u|S'_u, A_s)}{P(A'_u|A_s, B)} \sum_{S_u} P(S'_u|A_s, S_u)B_u(S_u)\right\} && (71)\\
&\quad \left\{\sum_{S_s} P(S'_s|S_s, S'_u, A_s, A'_u, B)P(S_s|A_s, A'_u, B)\right\} && (72)\\
&= \left\{\frac{P(A'_u|S'_u, A_s)}{P(A'_u|A_s, B)} \sum_{S_u} P(S'_u|A_s, S_u)B_u(S_u)\right\} && (73)\\
&\quad \left\{\sum_{S_s} \sum_{X'_i} P(S'_s|X'_i, S_s, A_s)P(X'_i|S'_u, A_s)B_s(S_s)\right\} && (74)
\end{aligned}
$$

Thus, the update of the belief state can be expressed, as for the transition matrix, in terms of the observation model, the user transition model, the domain model and the user output model.

If now the user state is assumed to be constant, the belief state update simplifies to

$$
B'(S') = \frac{P(A'_u|S'_u, A_s)}{P(A'_u|A_s, B)} \sum_{S_s} \sum_{X'_i} P(S'_s|X'_i, S_s, A_s)P(X'_i|S'_u, A_s)B_s(S_s) \quad (75)
$$

In terms of practical application of POMDP's to dialogue, relatively little has been done. An early attempt to show the value of the POMDP framework used a simplified vector representation of the belief state consisting of the most likely state, and the entropy of the belief state [39]. Later work by Zhang used a variable grid approximation to calculate value functions and the results suggested that this was significantly better than the entropy approximation[40, 41]. Note, however, that this work was based on simulations using Bayesian networks and hence the relevance to real dialogue systems remains an open question. Otherwise, little else has been done. The mathematical framework is enticing but the development of tractable optimisation algorithms remains for the moment elusive.

7 Response Generation

Generation is essentially the inverse of the understanding process described in section 3. Given the speech acts A_s and the embedded concepts C_s output by the dialogue manager, an acoustic signal $\boldsymbol{X}_o$ is required which conveys the intended meaning to the user in the most natural possible way, i.e. we seek

$$\arg\max_{\boldsymbol{X}_o} P(\boldsymbol{X}_o|A_s, C_s) = \arg\max_{\boldsymbol{X}_o} \left\{ \sum_W P(\boldsymbol{X}_o|W)P(W|A_s, C_s) \right\} \tag{76}$$

Again decoding sequentially leads to

$$\hat{W} = \arg\max_W \{P(W|A_s, C_s)\} \tag{77}$$

and

$$\hat{\boldsymbol{X}}_o = \arg\max_{\boldsymbol{X}_o} \left\{ P(\boldsymbol{X}_o|\hat{W}) \right\} \tag{78}$$

Equation 77 suggests that the domain specific models used in understanding (or a similar set trained on sample outputs rather than sample inputs) can be reused in "generation mode" to produce the most likely word sequence. However, the model $P(W|C)$ used for understanding was designed to extract the key content words and ignore syntax. Thus, a generation model based on this would lead to outputs with unacceptable surface structure. The solution to this is to smooth equation 77 with a separate language model trained on a large amount of well-formed data. Since this model is only required to capture surface structure, a simple N-gram will suffice. Equation 77 then becomes

$$\hat{W} = \arg\max_W \left\{ P(W|A_s, C_s) + \gamma \tilde{P}(W) \right\} \tag{79}$$

where γ controls the weight placed on grammatical well-formedness. In practice, the two stage process illustrated in Fig. 11 is used to implement equation 79. First $P(W|A_s, C_s)$ is used to generate a lattice of possible word sequences. $\tilde{P}(W)$ is then used to select the most likely single sequence from the lattice. Language generation based on this *overgenerate and filter* paradigm was first proposed by Langekilde and Knight in a system called Nitrogen [42, 43]. They used a hand-crafted semantic template grammar as the generator and a bigram language model for the filter. Atomic concepts in the template grammar were expanded using a lexicon and morphological expansion was performed by rule. However, since the model is tolerant to over-generation, the rules can be kept very simple.

The lack of an explicit tree-based representation of syntax can work well in narrow domains, but wider coverage benefits from more general syntactic knowledge. In [44], a tree-based model is introduced based on the UPenn XTAG grammar. This is claimed to allow richer output forms including the ability to model long-range effects such as the separation of parts of a collocation through embedded adjuncts.

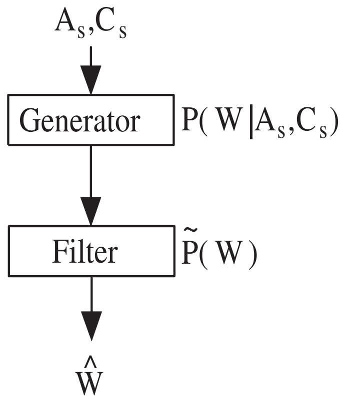

Figure 11: A Statistical Generation Model

None of the above systems pay much attention to sentence planning. In [45], a sentence planner called SPoT is described which operates on the overgenerate and filter paradigm. In this case the generator takes as input primitive dialogue acts, and stochastically combines these using a set of built-in combination operators such as merge, conjoin, relative-clause, etc. The filter is trained on a corpus of sentences graded by human judges to rank the generator output based on a number of features, the features themselves being determined automatically from the human-ranked data. On evaluation, the plans selected by SPoT are found to be statistically indistinguishable from the best plan selected by human judges.

Turning finally to the generation of the acoustic signal, speech synthesis systems implement equation 78 in essentially the same way as above. They have a large database of phone models and they attempt to find an expansion of $\hat{W}$ which meets various acoustic constraints. Most systems store the waveform segment corresponding to each phone model directly in the database making synthesis a trivial concatenation process [46]. Space precludes a detailed discussion of synthesis methods. However, within the theme of this paper, the work of Tokuda is noteworthy in that he has developed a system which uses HMMs directly as generators thus neatly implementing equation 78 directly [47].

8 Conclusions

This paper has attempted to provide a general statistical framework for spoken dialogue systems and a variety of on-going work has been described within that framework. There are clearly many hard problems to solve before this general holistic approach is ready for widespread deployment. However, the long-term pay-off is the ability to construct systems which are trainable from data, cheap to build, robust in operation and which can adapt their behaviour on-line.

References

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

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

[3] R Pieraccini, E Tzoukermann, Z Gorelov, E Levin, C-H Lee, and J-L Gauvain. Progress report on the chronus system: Atis benchmark results. In *Proc Speech and Natural Language Workshop*, pages 67–71, Harriman, New York, 1992.

[4] L Hirschman, M Bates, D Dahl, WM Fisher, J Garafolo, DS Pallet, C Pao, K Hunicke-Smith, P Price, and A Rudnicky. Multi-site data collection for a spoken language corpus. In *Proc Speech and Natural Language Workshop*, pages 7–14, Harriman, NY, 1992.

[5] L Hirschman, M Bates, D Dahl, WM Fisher, J Garafolo, DS Pallet, K Hunicke-Smith, P Price, A Rudnicky, and E Tzoukermann. Multi-site data collection and evaluation in spoken language understanding. In *Proc Workshop on Human Language Technology*, pages 19–24, Princeton, NJ, 1993.

[6] D Dahl, M Bates, M Brown, WM Fisher, K Hunicke-Smith, DS Pallet, C Pao, A Rudnicky, and E Shriberg. Expanding the scope of the atis task: the atis-3 corpus. In *Proc Human Language Technology Workshop*, pages 43–48, Plainsboro, NJ, 1994.

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

[8] K Lari and SJ Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4(1):35–56, 1990.

[9] S Fine, Y Singer, and N Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32:41–62, 1998.

[10] A Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201, 1995.

[11] K Lari and SJ Young. The application of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 5(3):237–258, 1991.

[12] CJ Leggetter and PC Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden markov models. *Computer Speech and Language*, 9(2):171–185, 1995.

[13] X Luo, S Roukos, and T Ward. Unsupervised adaptation of statistical parsers based on markov transform. In *IEEE Workshop Automatic Speech Recognition and Understanding*, pages 241–244, Colorado, USA, 1999.

[14] R Schwartz, S Miller, D Stallard, and J Makhoul. Language understanding using hidden understanding models. In *Proc Int Conf Spoken Language Processing*, Philadelphia, 1996.

[15] D Heckerman and E Horwitz. Inferring informational goals from free-text queries: a bayesian approach. In *Proc 14th Conf on Uncertainty in AI*, pages 230–238, 1998.

[16] HM Meng, W Lam, and C Wai. To believe is to understand. In *Eurospeech*, pages 2015–2018, Budapest, Hungary, 1999.

[17] HM Meng, W Lam, and KF Low. Learning belief networks for language understanding. In *IEEE Workshop on Automatic Speech Recognition and Understanding*, Keystone, Colorado, 1999.

[18] S Pulman. Conversational games, belief revision and bayesian networks. In *CLIN VII: 7th Computational Linguistics in the Netherlands meeting*, 1996.

[19] HM Meng, C Wai, and R Pieraccini. The use of belief networks for mixed-initiative dialog modelling. In *Int Conference on Spoken Language Processing*, Beijing, China, 2000.

[20] J Bellegarda and KEA Silverman. Toward unconstrained command and control: Data-driven semantic inference. In *Proc ICSLP*, pages 1258–1261, Beijing, China, 2000.

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

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

[23] RS Sutton and AG Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass, 1998.

[24] SJ Young. Probabilistic methods in spoken dialogue systems. *Philosophical Transactions of the Royal Society (Series A)*, 358(1769):1389–1402, 2000.

[25] MA Walker, JC Fromer, and S Narayanan. Learning optimal dialogue strategies: A case study of a spoken dialogue agent for email. In *Proceedings of ACL/COLING 98*, 1998.

[26] S Singh, DJ Litman, M Kearns, and M Walker. Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *Journal of Artificial Intelligence Research*, to appear, 2002.

[27] K Scheffler and SJ Young. Corpus-based dialogue simulation for automatic strategy learning and evaluation. In *Proc NAACL-2001 Workshop on Adaptation in Dialogue Systems*, Pittsburgh, USA, 2001.

[28] K Scheffler and SJ Young. Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. In *HLT 2002*, San Diego, USA, 2002.

[29] O Pietquin and S Renals. Asr system modelling for automatic evaluation and optimisation of dialogue systems. In *Int Conf Acoustics Speech and Signal Processing*, Florida, 2002.

[30] J Loch and S Singh. Using eligibiliby traces to find the best memoryless policy in partially observable markov decision processes. In *15th In Conf on Machine Learning (ICML-98)*, 1998.

[31] DP Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, 1987.

[32] EJ Sondik. *The Optimal Control of Partially Observable Markov Decision Processes*. Phd, Stanford University, 1971.

[33] R Parr and S Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the IJCAI*, 1995.

[34] ML Littman, AR Cassandra, and LP Kaelbling. Learning policies for partially observable environments: Scaling up. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370, San Francisco, CA, 1995. Morgan Kaufmann.

[35] AR Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, 1997.

[36] LP Kaelbling, ML Littman, and AR Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[37] M Hauskrecht. Value-function approximations for partially observable markov decision problems. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[38] WD Smart and LP Kaelbling. Practical reinforcement learning in continuous spaces. In *Proc 17th Int Conf on Machine Learning*, 2000.

[39] N Roy, J Pineau, and S Thrun. Spoken dialogue management using probabilistic reasoning. In *Proceedings of the ACL 2000*, 2000.

[40] B Zhang, Q Cai, J Mao, and B Guo. Planning and acting under uncertainty: A new model for spoken dialogue system. In *Proc 17th Conf on Uncertainty in AI*, Seattle, 2001.

[41] B Zhang, Q Cai, J Mao, E Chang, and B Guo. Spoken dialogue management as planning and acting under uncertainty. In *Eurospeech*, Aalborg, Denmark, 2001.

[42] I Langkilde and K Knight. Generation that exploits corpus-based statistical knowledge. In *Proc 17th Int Conf on Computational Linguistics (COLING-ACL 1998)*, Montreal, Canada, 1998.

[43] I Langkilde and K Knight. The practical value of n-grams in generation. In *Proc 9th Int Natural Language Workshop (INLG 98)*, Niagara-on-the-Lake, Canada, 1998.

[44] S Bangalore and OC Rambow. Exploiting a probabilistic hierarchical model for generation. In *Proc 18th Int Conf on Computational Linguistics (COLING 2000)*, Saarbrucken, Germany, 2000.

[45] M Walker, OC Rambow, and M Rogati. Training a sentence planner for spoken dialogue using boosting. *Computer Speech and Language*, 16(3-4), 2002.

[46] AW Black and P Taylor. Automatically clustering similar units for unit selection speech synthesis. In *Proc Eurospeech*, pages 601–604, Rhodes,Greece, 1997.

[47] Y Tokuda, T Yoshimura, T Masuko, T Kobayashi, and T Kitamura. Speech parameter generation algorithms for hmm-based speech synthesis. In *Proc Int Conf Acoustics Speech and Signal Processing*, pages 1315–1318, Istanbul, Turkey, 2000.