Statistical User Simulation with a Hidden Agenda

Jost Schatzmann and Blaise Thomson and Steve Young
Cambridge University Engineering Department
Trumpington Street, Cambridge CB2 1PZ, United Kingdom
{js532,brmt2,sjy}@eng.cam.ac.uk

Abstract

Recent work in the area of probabilistic user simulation for training statistical dialogue managers has investigated a new *agenda*-based user model and presented preliminary experiments with a handcrafted model parameter set. Training the model on dialogue data is an important next step, but non-trivial since the user agenda states are not observable in data and the space of possible states and state transitions is intractably large. This paper presents a summary-space mapping which greatly reduces the number of state transitions and introduces a tree-based method for representing the space of possible agenda state sequences. Treating the user agenda as a hidden variable, the forward/backward algorithm can then be successfully applied to iteratively estimate the model parameters on dialogue data.

1 Introduction

1.1 Statistical user simulation

A key advantage of taking a statistical approach to dialogue manager (DM) design is the ability to formalise design criteria as objective reward functions and to learn an optimal dialogue policy from human-computer dialogue data (Young, 2002). The amount of suitably annotated in-domain data required for training a statistical system, however, typically exceeds the size of available dialogue corpora by several orders of magnitude and it is thus common practise to use a two-phased simulation-based approach. First, a statistical model of user behaviour is trained on the limited amount of available data. The trained model is then used to simulate any number of dialogues with the interactively learning dialogue manager (Levin et al., 2000; Scheffler and Young, 2002; Pietquin, 2004; Georgila et al., 2005; Lemon et al., 2006; Rieser and Lemon, 2006; Schatzmann et al., 2006).

1.2 Agenda-based user modelling

Recent work by Schatzmann et al. (2007) has presented a new technique for user simulation based on explicit representations of the *user goal* and the *user agenda*, which provide compact models of the dialogue context and the user's "state of mind" and are dynamically updated during the dialogue. Experimental results with the statistical POMDP-based Hidden Information State dialogue system (Young et al., 2007; Thomson et al., 2007) show that a competitive dialogue policy can be learnt even with handcrafted user model parameters.

1.3 Training on real data

While this result is useful for bootstrapping a prototype DM when no access to dialogue data is available, training the agenda-model on real human-computer dialogue data is an important next step. Training avoids the effort and expertise needed to manually set the model parameters and ensures that the learned system policy is optimized for human dialogue behaviour rather than the handcrafted simulator. The implementation of a suitable training algorithm for the agenda-based user model, however, is non-trivial since the user agenda and goal states are not observable in data. Moreover, the space of possible states and state transitions is intractably large.

1.4 Paper overview

This paper reviews the agenda-based user model (Section 2) and presents an Expectation-Maximization (EM)-based training method (Section 3) which models the observable dialogue data in terms of a sequence of hidden user states. Section 4 discusses the tractability problems associated with the vast state space and suggests a summary-space mapping for state transitions. Using an efficient tree-based method for generating state sequences on-the-fly, the forward/backward algorithm can then be applied to iteratively estimate the model parameters on data. Section 5 concludes with a brief evalution.

2 Agenda-based user simulation

2.1 User simulation at a semantic level

The agenda-based model introduced by Schatzmann et al. (2007) formalises human-machine dialogue at a semantic level as a sequence of states and dialogue acts[1]. At any time t, the user is in a state S, takes action a_u, transitions into the intermediate state S', receives machine action a_m, and transitions into the next state S'' where the cycle restarts.

$$S \rightarrow a_u \rightarrow S' \rightarrow a_m \rightarrow S'' \rightarrow \cdots \tag{1}$$

Assuming a Markovian state representation, user behaviour can be decomposed into three models: $P(a_u|S)$ for action selection, $P(S'|a_u,S)$ for the state transition into S', and $P(S''|a_m,S')$ for the transition into S''. Dialogue acts are assumed to be of the form *act(a=x, b=y,...)*, where *act* denotes the type of action (such as *hello*, *inform* or *request*) and act items *a=x* and *b=y* denote slot-value pairs, such as *food=Chinese* or *stars=5* as described in (Young et al., 2005).

2.2 State decomposition into goal and agenda

Inspired by agenda-based approaches to dialogue management (Wei and Rudnicky, 1999; Lemon et al., 2001; Bohus and Rudnicky, 2003) the user state is factored into an agenda A and a goal G.

$$S = (A, G) \qquad \text{and} \qquad G = (C, R) \tag{2}$$

During the course of the dialogue, the goal G ensures that the user behaves in a consistent, goal-directed manner. G consists of constraints C which specify the required venue, eg. “a centrally located bar serving beer”, and requests R which specify the desired pieces of information, eg. “the name, address and phone number of the venue”.

The user agenda A is a stack-like structure containing the pending user dialogue acts that are needed to elicit the information specified in the goal. At the start of the dialogue a new goal is randomly generated using the system database and the agenda is populated by converting all goal constraints into *inform* acts and all goal requests into *request* acts. A *bye* act is added at the bottom of the agenda to close the dialogue (cf. Fig. 5 in the Appendix.).

As the dialogue progresses the agenda is dynamically updated and acts are selected from the top of the agenda to form user acts a_u. In response to incoming machine acts a_m, new user acts are pushed onto the agenda and no longer relevant ones are removed. The agenda thus serves as a convenient way of tracking the progress of the dialogue as well as encoding the relevant dialogue history. Dialogue acts can also be temporarily stored when actions of higher priority need to be issued first, hence providing the simulator with a simple model of user memory (see Fig. 5 for an illustration). When using an n-gram based approach, by comparison, such long-distance dependencies between dialogue turns are neglected unless n is set to a large value, which in turn often leads to poor model parameters estimates.

Another, perhaps less obvious, advantage of the agenda-based approach is that it enables the simulated user to take the initiative when the dialogue is corrupted by recognition errors or when the incoming system action is not relevant to the current task. The latter point is critical for training statistical dialogue managers because policies are typically learned from a random start. The “dialogue history” during the early training phase is thus often a sequence of random dialogue acts or dialogue states that has never been seen in the training data. The stack of dialogue acts on the agenda enables the user model to take the initiative in such cases and behave in a goal-directed manner even if the system is not.

2.3 Action selection and state transition models

As explained in detail in (Schatzmann et al., 2007), the decomposition of the user state S into a goal G and an agenda A simplifies the models for action selection and state transition. Since the agenda (of length N) is ordered according to priority, with $A[N]$ denoting the top and $A[1]$ denoting the bottom item, forming a user response is equivalent to popping n items of the top of the stack. Using $A[N-n+1..N]$ as a Matlab-like shorthand notation for the top n items on A, the action selection model can be expressed as

$$P(a_u|S) = \delta(a_u, A[N- n+1..N])P(n|A,G) \tag{3}$$

where $\delta(p,q)$ is 1 iff $p = q$ and zero otherwise.

The state transition models $P(S'|a_u,S)$ and $P(S''|a_m,S')$ are rewritten as follows. Letting A' denote the agenda after popping off a_u and using $N' = N - n$ to denote the size of A', we have

$$A'[i] \quad := \quad A[i] \quad \forall i \in [1..N']. \tag{4}$$

Using this definition of A' and assuming that the goal remains constant when the user executes a_u, the first state transition depending on a_u is entirely deterministic:

$$\begin{aligned} P(S'|a_u,S) &= P(A',G'|a_u,A,G) \\ &= \delta(A',A[1..N'])\delta(G',G). \end{aligned} \tag{5}$$

The second state transition based on a_m can be decomposed into *goal update* and *agenda update* modules:

$$\begin{aligned} &P(S''|a_m,S') \\ &\quad = \underbrace{P(A''|a_m,A',G'')}_{\text{agenda update}}\underbrace{P(G''|a_m,G')}_{\text{goal update}}. \end{aligned} \tag{6}$$

[1] The terms *dialogue act* and *dialogue action* are used interchangeably here.

3 Model Parameter Estimation

3.1 The user state as a hidden variable

Estimating the parameters of the action selection and state transition models is non-trivial, since the goal and agenda states are not observable in training data.

Previous work on the state-based approach to statistical user simulation (Georgila et al., 2005; Lemon et al., 2006; Rieser and Lemon, 2006) has circumvented this problem by annotating training data with dialogue state information and conditioning user output on the observable dialogue state rather than the unobservable user state. While this simplifies the training process, providing the necessary annotation requires a considerable effort. If done manually, the process is often expensive and it can be difficult to ensure inter-annotator agreement. Using an automatic tool for dialogue state annotation (Georgila et al., 2005) can improve efficiency, but the development of the tool itself is a time-consuming process.

The parameter estimation approach presented here avoids the need for dialogue state annotation by modelling the observable user and machine dialogue acts in terms of a *hidden* sequence of agendas and user goal states. More formally, the dialogue data $\mathcal{D}$ containing dialogue turns 1 to T

$$\mathcal{D} = \{\mathbf{a_u}, \mathbf{a_m}\} = \{a_{m,1}, a_{u,1} ..., a_{m,T}, a_{u,T}\} \qquad (7)$$

is modelled in terms of latent variables

$$X = \{\mathbf{A}, \mathbf{G}\} \qquad (8)$$

where

$$\begin{aligned} \mathbf{A} &= \{A_1, A'_1, ..., A_T, A'_T\} && (9) \\ \mathbf{G} &= \{G_1, G'_1, ..., G_T, G'_T\}. && (10) \end{aligned}$$

Collecting the results from Section 2, and noting that from (5) the choice of n deterministically fixes A', the joint probability can hence be expressed as

$$P(X, D) = P(\mathbf{A}, \mathbf{G}, \mathbf{a_u}, \mathbf{a_m}) =$$

$$\prod_{t=1}^{T} P(n_t|A_t, G_t) P(A''_t|a_{m,t}, A'_t, G''_t) P(G''_t|a_{m,t}, G'_t). \qquad (11)$$

The goal is to learn maximum likelihood (ML) values for the model parameter set θ such that the log likelihood

$$\mathcal{L}(\theta) = \log P(\mathcal{D}|\theta) = \log \sum_X P(X, \mathcal{D}|\theta) \qquad (12)$$

is maximized

$$\theta_{ML} = \arg\max_\theta \mathcal{L}(\theta). \qquad (13)$$

3.2 An EM-based approach

The direct optimization of $\mathcal{L}(\theta)$ is not possible, however, an iterative Expectation-Maximization (EM)-based approach (Dempster et al., 1977) can be used to find a (local) maximum of the latent variable model likelihood. Using Jensen's inequality, any distribution $q(X)$ can be used to obtain a lower bound on $\mathcal{L}(\theta)$

$$\begin{aligned} &\mathcal{L}(\theta) = \\ &\log \sum_X q(X) \frac{P(X, \mathcal{D}|\theta)}{q(X)} \geq \sum_X q(X) \log \frac{P(X, \mathcal{D}|\theta)}{q(X)} \\ &\qquad \stackrel{\text{def}}{=} \mathcal{F}(q(X), \theta). \qquad (14) \end{aligned}$$

Since $\mathcal{L}(\theta)$ is always greater or equal to the "negative free energy" $\mathcal{F}(q(X), \theta)$ the problem of maximizing $\mathcal{L}(\theta)$ is equivalent to maximizing $\mathcal{F}(q(X), \theta)$. Starting from arbitrarily selected model parameters, EM iterates by alternating an E-step and an M-step.

During the E-step, the distribution $q^{(k)}(X)$ over the latent variables is estimated for fixed model parameters $\theta^{(k-1)}$

$$q^{(k)}(X) := \arg\max_{q(X)} \mathcal{F}(q(X), \theta^{(k-1)}). \qquad (15)$$

It can be shown that this is achieved by setting

$$q^{(k)}(X) = P(X|\mathcal{D}, \theta^{(k-1)}). \qquad (16)$$

Using Bayes rule and the law of total probability the RHS of Eq. 16 can be expressed as

$$\begin{aligned} &P(X|\mathcal{D}, \theta^{(k-1)}) \\ &= \frac{P(\mathcal{D}|X, \theta^{(k-1)}) P(X|\theta^{(k-1)})}{\sum_X P(\mathcal{D}|X, \theta^{(k-1)}) P(X|\theta^{(k-1)})}. \qquad (17) \end{aligned}$$

Resubstituting (7) and (8) into (17) completes the E-step:

$$\begin{aligned} &q^{(k)}(\mathbf{A}, \mathbf{G}) \\ &= \frac{P(\mathbf{a_u}, \mathbf{a_m}|\mathbf{A}, \mathbf{G}, \theta^{(k-1)}) P(\mathbf{A}, \mathbf{G}|\theta^{(k-1)})}{\sum_{\mathbf{A},\mathbf{G}} P(\mathbf{a_u}, \mathbf{a_m}|\mathbf{A}, \mathbf{G}, \theta^{(k-1)}) P(\mathbf{A}, \mathbf{G}|\theta^{(k-1)})}. \\ &\qquad (18) \end{aligned}$$

The M-step now optimizes $\mathcal{F}(q(X), \theta)$ with respect to θ whilst holding $q^{(k)}(X)$ fixed

$$\theta^{(t)} := \arg\max_\theta \mathcal{F}(q^{(k)}(X), \theta). \qquad (19)$$

This is achieved by maximizing the auxiliary function

$$Q(\theta, \theta^{(k-1)}) = \sum_X P(X, \mathcal{D}|\theta^{(k-1)}) \log P(X, \mathcal{D}|\theta). \qquad (20)$$

Substituting Eq. 11 into the above, differentiating with respect to θ and setting the result to zero, one arrives at the parameter reestimation formulae shown in Eqs. 21-23 in Fig. 1.

$$\hat{P}(n|A,G) = \frac{\sum_t P(A_t = A, G_t = G|\mathbf{a_u}, \mathbf{a_m}, \theta^{(k-1)})\delta(n_t, n)}{\sum_t P(A_t = A, G_t = G|\mathbf{a_u}, \mathbf{a_m}, \theta^{(k-1)})} \tag{21}$$

$$\hat{P}(A''|a_m, A', G'') = \frac{\sum_t P(A''_t = A'', A'_t = A', G''_t = G''|\mathbf{a_u}, \mathbf{a_m}, \theta^{(k-1)})\delta(a_{m,t}, a_m)}{\sum_t P(A'_t = A', G''_t = G''|\mathbf{a_u}, \mathbf{a_m}, \theta^{(k-1)})\delta(a_{m,t}, a_m)} \tag{22}$$

$$\hat{P}(G''|a_m, G') = \frac{\sum_t P(G''_t = G'', G'_t = G'|\mathbf{a_u}, \mathbf{a_m}, \theta^{(k-1)})\delta(a_{m,t}, a_m)}{\sum_t P(G'_t = G'|\mathbf{a_u}, \mathbf{a_m}, \theta^{(k-1)})\delta(a_{m,t}, a_m)} \tag{23}$$

Figure 1: Model parameter update equations for the action selection and agenda and goal state transition models. Note that $\delta(n_t, n)$ is one iff $n_t = n$ and zero otherwise. Similarly, $\delta(a_{m,t}, a_m)$ is one iff $a_{m,t} = a_m$ and zero otherwise.

4 Implementation

4.1 Tractability considerations

In the Hidden Information State (HIS) Dialogue System (Young et al., 2007) used for the experiments presented in this paper, the size of the user and machine dialogue action sets $\mathcal{U}$ and $\mathcal{M}$ is

$$|\mathcal{U}| \approx 10^3 \qquad \text{and} \qquad |\mathcal{M}| \approx 10^3. \tag{24}$$

Goals are composed of N_C constraints taken from the set of constraints $\mathcal{C}$, and N_R requests taken from the set of requests $\mathcal{R}$. Note that the ordering of constraints and requests does not matter, and there are no duplicate constraints or requests. Using typical values for goal specifications during previous HIS Dialogue System user trials (Thomson et al., 2007) the size of the goal state space can be estimated as

$$|\mathcal{G}| = \binom{|\mathcal{C}|}{N_C}\binom{|\mathcal{R}|}{N_R} = \binom{50}{4}\binom{8}{3} \approx 10^7. \tag{25}$$

The size of the agenda state space $\mathcal{A}$ depends on the number of unique user dialogue acts $|\mathcal{U}|$ as defined above and the maximum number N_A of user dialogue acts on the agenda. The maximum length of the agenda is a design choice, but it is difficult to simulate realistic dialogues unless it is set to at least $N_A = 8$. If fully populated, $\mathcal{A}$ therefore comprises the vast number of

$$|\mathcal{A}| = \frac{|\mathcal{U}|!}{(|\mathcal{U}| - N_A)!} \approx 10^{20}. \tag{26}$$

potential agenda states[2] and the number of parameters needed to model $P(A''|a_m, A', G'')$ is of the order

$$|\mathcal{A} \times \mathcal{M} \times \mathcal{A} \times \mathcal{G}| \approx 10^{50}. \tag{27}$$

4.2 Agenda updates as a sequence of push actions

The estimates show that when no restrictions are placed on A'', the space of possible state transitions is vast. It can however be assumed that A'' is derived from A' and that each transition entails only a limited number of well-defined atomic operations (Schatzmann et al., 2007).

More specifically, the agenda transition from A' to A'' can be viewed as a sequence of push-operations in which dialogue acts are added to the top of the agenda. In a second ”clean-up” step, duplicate dialogue acts, “empty” acts, and unnecessary *request()* acts for already filled goal request slots must be removed but this is a deterministic procedure so that it can be excluded in the following derivation for simplicity. Considering only the push-operations, the items 1 to N' at the bottom of the agenda remain fixed and the update model is rewritten as follows:

$$\begin{aligned} &P(A''|a_m, A', G'') \\ &\quad = P(A''[1..N'], A''[N'{+}1..N'']|a_m, A'[1..N'], G'') \\ &\quad = \delta(A''[1..N'], A'[1..N']) \\ &\qquad \cdot \; P(A''[N'{+}1..N'']|a_m, G''). \end{aligned} \tag{28}$$

The second term on the RHS of Eq. 28 can now be further simplified by assuming that every dialogue act item (slot-value pair) in a_m triggers one push-operation. This assumption can be made without loss of generality, because it is possible to push an “empty” act (which is later removed) or to push an act with more than one item. The advantage of this assumption is that the known number M of items in a_m now determines the number of push-operations. Hence $N'' = N' + M$ and

$$\begin{aligned} &P(A''[N'{+}1..N'']|a_m, G'') \\ &\quad = P(A''[N'{+}1..N'{+}M]|a_m[1..M], G'') \qquad (29) \\ &\quad = \prod_{i=1}^{M} P(\underbrace{A''[N'{+}i]}_{a_{push}} | \underbrace{a_m[i]}_{a_{cond}}, G'') \qquad (30) \end{aligned}$$

The expression in Eq. 30 shows that each item $a_m[i]$ in the system act triggers one push operation, and that this

[2] Note that the order of agenda items matters and that there are no duplicate items.

operation is conditioned on the goal. For example, given that the item *x=y* in $a_m[i]$ violates the constraints in G'', one of the following might be pushed onto A'': *negate(), inform(x=z), deny(x=y, x=z),* etc.

Let $a_{push} \in \mathcal{U}$ denote the pushed act $A''[N' + i]$ and $a_{cond} \in \mathcal{M}$ denote the conditioning dialogue act containing the single dialogue act item $a_m[i]$. Omitting the Dirac delta function in Eq. 28, the agenda update step then reduces to the repeated application of a *push transition model* $P(a_{push}|a_{cond}, G'')$. The number of parameters needed to model $P(a_{push}|a_{cond}, G'')$ is of the order

$$|\mathcal{U} \times \mathcal{M} \times \mathcal{G}| \approx 10^{13}. \quad (31)$$

While still large, this number is significantly smaller then the number of parameters needed to model unrestricted transitions from A' to A'' (cf. Eq. 27).

4.3 A summary space model for push transitions

To further reduce the size of the model parameter set and make the estimation of $P(a_{push}|a_{cond}, G'')$ tractable, it is useful to introduce the concept of a "summary space", as has been previously done in the context of dialogue management (Williams and Young, 2005). First, a function ϕ is defined for mapping the machine dialogue act $a_{cond} \in \mathcal{M}$ and the goal state $G'' \in \mathcal{G}$ from the space of machine acts $\mathcal{M}$ and goal states $\mathcal{G}$ to a smaller summary space Z_{cond} of "summary conditions"

$$\phi : \mathcal{M} \times \mathcal{G} \mapsto Z_{cond} \quad \text{with} \quad |\mathcal{M} \times \mathcal{G}| \gg |Z_{cond}|. \quad (32)$$

Secondly, a "summary push action" space Z_{push} is defined, which groups real user dialogue acts into a smaller set of equivalence classes. Using a function ω, summary push actions are mapped back to "real" dialogue acts

$$\omega : Z_{push} \mapsto \mathcal{U} \quad \text{with} \quad |Z_{push}| \ll |\mathcal{U}|. \quad (33)$$

Agenda state transitions can now be modelled in summary space using

$$P(a_{push}|a_{cond}, G'') \approx P(z_{push}|z_{cond}) \quad (34)$$

where $z_{push} \in Z_{push}$ and $z_{cond} \in Z_{cond}$ and

$$\begin{aligned} z_{cond} &= \phi(a_{cond}, G'') &(35)\\ a_{push} &= \omega(z_{push}). &(36) \end{aligned}$$

For the experiments presented in this paper, 20 summary conditions and 20 summary push actions were defined, with examples shown in Fig 6. The total number of parameters needed to model $P(z_{push}|z_{cond})$ is therefore

$$|Z_{cond} \times Z_{push}| = 400. \quad (37)$$

The parameter set needed to model agenda transitions is now small enough to be estimated on real dialogue data.

4.4 Representing agenda state sequences

Given our estimate of $|\mathcal{A}| \approx 10^{20}$ for the size of the agenda state space, the direct enumeration of all states in advance is clearly intractable. The actual number of states needed to model a particular dialogue act sequence, however, is much smaller, since agenda transitions are restricted to push/pop operations and conditioned on dialogue context. The training algorithm can exploit this by generating state-sequences on-the-fly, and discarding any state sequence X for which $P(X, \mathcal{D}|\theta) = 0$.

A suitable implementation for this is found in the form of a dynamically growing agenda-tree, which allows agenda-states to be represented as tree-nodes and state transitions as branches. The tree is initialised by creating a root node containing an empty agenda and then populating the agenda according to the goal specification as explained in Sect. 2. However, since the initial ordering of dialogue acts on the agenda is unknown, all possible permutations of constraints and requests must be created, resulting in a row of $N_C! \cdot N_R!$ initial agendas (cf. Fig. 2).

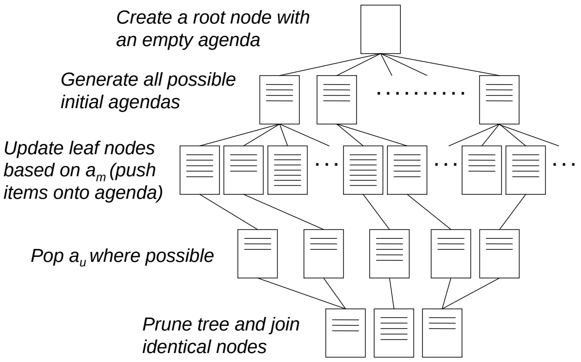
Create a root node with an empty agenda
Generate all possible initial agendas
Update leaf nodes based on a_m (push items onto agenda)
Pop a_u where possible
Prune tree and join identical nodes

Figure 2: Tree-based method for representing state sequences.

4.4.1 Updating the tree based on a_m

The dialogue is now "parsed" by growing the tree and creating branches for all possible state sequences. Updates based on a machine dialogue act a_m involve mapping each item in a_m to its corresponding summary condition z_{cond} using the function ϕ. For each z_{cond} a list of summary push actions z_{push} is generated, discarding cases where $P(z_{push}|z_{cond}) = 0$. The summary push actions are then mapped back to real push actions using ω and used to create new agendas which are attached to the tree as new branches. The probability of the transition/branch is computed as the product of the probabilities of the real push actions. (See Fig. 6 in the appendix for a detailed illustration.)

The leaf nodes are now cleaned up in a deterministic procedure to remove empty and duplicate dialogue acts, to delete all dialogue acts below a *bye()* act, and to remove all requests for items that have already been filled in the user goal. (An exception to the latter is made for requests that have just been added to the agenda, such that the simulated user can re-request filled items.)

4.4.2 Updating the tree based on a_u

In the next step, the tree is updated based on the observed user act a_u. This part simplifies to popping a_u from the top of the agenda wherever this is possible. Agendas which do not allow a_u to be popped off represent states with zero probability and can be discarded. In all other cases, a new node with the updated agenda is attached to the tree. The branch is marked as a pop-transition and its probability is computed based on the number of items popped.

4.4.3 Pruning the tree and joining identical nodes

Once the update based on a_u is completed, the tree is pruned to reduce the number of nodes and branches. First, all branches which were not extended during the dialogue turn, i.e. branches where a_u could not be popped off the leaf node agenda, are removed. All remaining branches represent possible sequences of agenda states with non-zero probability for the dialogue acts seen so far. In a second step, a more aggressive type of pruning can be carried out by removing all branches which do not have a given minimum leaf node probability. After pruning, the size of the tree is further reduced by joining nodes with identical agendas.

4.5 Action selection and goal update model

The action selection and goal update models experience similar tractability problems as the agenda update model, but in both cases a straightforward solution was found to produce satisfactory results. To simplify the action selection model $P(n|A,G)$, the random variable n can be assumed independent of A and G. The probability distribution $P(n)$ over small integer values for n (typically in the range from 0 to 6) can then be estimated directly from dialogue data by obtaining frequency counts of the number of dialogue act items in every user act.

The goal update model $P(G''|a_m,G')$ is decomposed into separate update steps for the constraints and requests. Assuming that R'' is conditionally independent of C' given C'' it is easy to show that

$$\begin{aligned} & P(G''|a_m,G') \\ & \quad = P(R''|a_m,R',C'')P(C''|a_m,R',C'). \end{aligned} \tag{38}$$

The two update steps can be treated separately and implemented deterministically using two rules: 1) If R' contains an empty slot *u* and a_m is a dialogue act of the form *inform(u=v,r=s,...)*, then R'' is derived from R' by setting *u=v* given that no other information in a_m violates any constraints in C''. 2) If a_m contains a request for the slot *x*, a new constraint *x=y* is added to C' to form C''. The latter does not imply that the user necessarily responds to a system request for any slot *x*, since the agenda update model does not enforce a corresponding user dialogue act to be issued.

4.6 Applying the forward/backward algorithm

Using the summary space mapping for agenda transitions and simplifying assumptions for the goal update and action selection model, the parameter update equation set reduces to a single equation:

$$\begin{aligned} & \hat{P}(z_{push}|z_{cond}) = \\ & \qquad \frac{\sum_k P(z_{push,k} = z_{push}, z_{cond,k} = z_{cond}|\mathbf{a_u},\mathbf{a_m},\theta)}{\sum_k P(z_{cond,k} = z_{cond}|\mathbf{a_u},\mathbf{a_m},\theta)} \end{aligned} \tag{39}$$

Note that k is used here rather than t, since every dialogue turn t involves two state transitions, and there are hence $K = 2T$ observations and update steps.

The parameter update equation can now be efficiently implemented by applying the forward/backward algorithm. Let $\alpha_i(k)$ denote the forward probability of being in state i after seeing the observations from 1 to k, and let $\beta_i(k)$ denote the backward probability of seeing the observations from $k+1$ to K, given that we are in state i after update step k:

$$\alpha_i(k) = P(o_1, o_2, \ldots, o_k, x_k = i|\theta) \tag{40}$$

$$\beta_i(k) = P(o_{k+1}, o_{k+2}, \ldots, o_K|x_k = i, \theta) \tag{41}$$

Based on the observations, a tree of agendas is constructed as described in Section 4.4. After the last observation K, all agenda items have been popped, so that the leaf node agendas are empty and can be merged to form a single end node. The forward/backward probabilities are now initialised using

$$\alpha_i(1) = \frac{1}{N_C!N_R!} \quad , \quad 1 \leq i \leq N_C!N_R! \tag{42}$$

$$\beta_{end}(K) = 1 \tag{43}$$

and then recursively defined for the update steps from $k = 2$ to $k = K - 1$ using

$$\alpha_j(k) = \sum_i \alpha_i(k-1)a_{ij} \tag{44}$$

$$\beta_i(k) = \sum_j a_{ij}\beta_j(k+1) \tag{45}$$

where the transition probability a_{ij} of transitioning from state i to j depends on whether it is a push or a pop transition. When the transition involves popping n items off the agenda, a_{ij} equals $P(n)$. If the transition involves a sequence of push actions, then a_{ij} is defined as the product of the probability of the associated real push actions (see Fig. 6 in the appendix for an illustration).

Using the forward/backward probabilities, one can now compute the probability $\tau_k(i,j)$ of transitioning from state i to state j at update step k as

$$\tau_k(i,j) = \frac{\alpha_i(k) a_{ij} \beta_j(k+1)}{\alpha_{end}(K)}. \qquad (46)$$

Finally, the push transition model parameters are updated using

$$\hat{P}(z_{push}|z_{cond}) = \frac{\sum_{\{k,i,j|SPA=z_{push},SC=z_{cond}\}} \tau_k(i,j)}{\sum_{\{k,i,j|SC=z_{cond}\}} \tau_t(i,j)} \qquad (47)$$

where the summation subscripts indicate if the summary push action (SPA) z_{push} and summary condition (SC) z_{cond} were used to transition from i to j at step k.

5 Evaluation

5.1 Dialogue training data

The parameter estimation approach presented in this paper was tested using a small corpus collected with the HIS Dialogue System (Young et al., 2007; Thomson et al., 2007; Schatzmann et al., 2007). The dataset consists of 160 dialogues from the tourist information domain, recorded with 40 different speakers, each of whom completed 4 dialogues. In total, the corpus contains 6452 dialogue turns and 21667 words. All utterances were manually transcribed and annotated using the set of dialogue act definitions described in Section 2.1. No dialogue state or user state annotation was needed.

5.2 Training results

The user model was trained on the dialogue corpus described above and Fig. 3 shows the number of agenda tree leaf nodes during a typical training episode on a sample dialogue. For each machine dialogue act, the tree is extended and 1 or more new nodes are attached to each tree branch, so that the number of leaf nodes stays constant or increases. Pop operations are then performed where possible, the tree is pruned and identical nodes are joined so that the number stays constant or decreases. At the end of the dialogue, only a single leaf node with an empty agenda remains.

When plotting the log probability of the data (Fig. 4), it can be seen that the EM-based algorithm produces a monotonically increasing curve (as expected). The algorithm quickly converges to a (local) optimum, so that in practise only a few iterations are needed. For illustration purposes, the training run in Fig. 4 was performed on two dialogues. As can be seen the log prob of the individual dialogues increases (top two lines), just as the log prob of the complete dataset (bottom line).

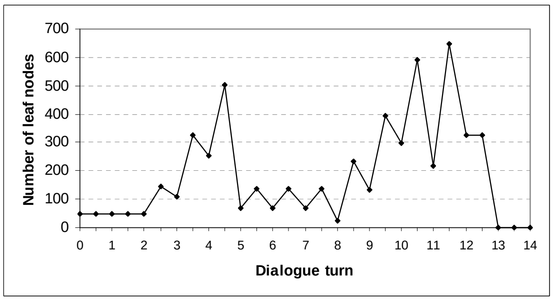

Figure 3: Graph showing the number of agenda tree leaf nodes after each observation during a training run performed on a single dialogue.

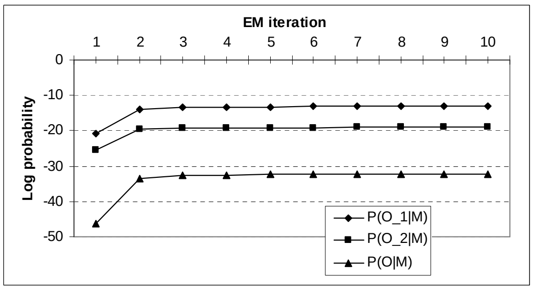

Figure 4: Graph showing a monotonous increase in log probability $\mathcal{L}(\theta)$ after each iteration of the EM algorithm.

5.3 Comparison of real and simulated data

An initial evaluation of the simulation quality has been performed by testing the similarity between real and simulated data. Table 1 shows basic statistical properties of dialogues collected with 1) real users, 2) the trained agenda model and 3) the handcrafted baseline simulator used by Schatzmann et al. (2007). All results were obtained with the same trained dialogue manager and the same set of user goal specifications. Since the model aims to reproduce user behaviour but not recognition errors, only the subset of 84 dialogues with a semantic accuracy above 90% was used from the real dialogue corpus[3]. The results show that the trained simulator performs better than the handcrafted baseline. The difference between the statistical properties of dialogues generated with the trained user model and those collected with real users is not statistically significant with confidence of more than 95%. Hence, based on these metrics, the trained agenda model appears to more closely match real human dialogue behaviour. One may expect that a dialogue system trained on this model is likely to perform better on real users than a system trained with the handcrafted simulator, but this is still an open research question.

[3] Semantic accuracy was measured in terms of substitution, insertion and deletion errors as defined by Boros et al. (1996).

	Real Users	Tr. Sim	Hdc. Sim
Sample size	84	1000	1000
Dial. length	3.30±0.53	3.38±0.07	4.04±0.19
Compl. rate	0.98±0.03	0.94±0.02	0.93±0.02
Performance	16.23±1.01	15.32±0.34	14.65±0.50

Table 1: Comparison of basic statistical properties of real and simulated dialogue data (mean±95% confidence thresholds). Dialogue length is measured in turns, task completion rate is based on the recommendation of a correct venue, and dialogue performance is computed by assigning a 20 point reward for a successful recommendation (0 otherwise) and subtracting 1 point for every turn.

6 Summary

This paper has extended recent work on an agenda-based user model for training statistical dialogue managers and presented a method for estimating the model parameters on human-computer dialogue data. The approach models the observable dialogue acts in terms of a sequence of hidden user states and uses an EM-based algorithm to iteratively estimate (locally) optimal parameter values.

In order to make estimation tractable, the training algorithm is implemented using a summary-space mapping for state transitions. Agenda state sequences are represented using tree structures, which are generated on-the-fly for each dialogue in the training corpus. Experimental results show that the forward/backward algorithm can be successfully applied to recompute the model parameters.

A comparison of real and simulated dialogue data has shown that the trained user model outperforms a handcrafted simulator and produces dialogues that closely match statistical properties of real data. While these initial results are promising, further work is needed to refine the summary state mapping and to fully evaluate the trained model. We look forward to reporting these results in a future paper.

References

D. Bohus and A. Rudnicky. 2003. Ravenclaw: Dialog management using hierarchical task decomposition and an expectation agenda. In *Proc. of Eurospeech*. Geneva, Switzerland.

M. Boros, W. Eckert, F. Gallwitz, G. Gorz, G. Hanrieder, and H. Niemann. 1996. Towards understanding spontaneous speech: Word accuracy vs. concept accuracy. In *Proc. of ICSLP*. Philadelphia, PA.

A. Dempster, N. Laird, and D. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38.

K. Georgila, J. Henderson, and O. Lemon. 2005. Learning user simulations for information state update dialog systems. In *Proc. of Eurospeech*. Lisbon, Portugal.

O. Lemon, A. Bracy, A. Gruenstein, and S. Peters. 2001. The WITAS multi-modal dialogue system I. In *Proc. of Eurospeech*. Aalborg, Denmark.

O. Lemon, K. Georgila, and J. Henderson. 2006. Evaluating Effectiveness and Portability of Reinforcement Learned Dialogue Strategies with real users: the TALK TownInfo Eval. In *Proc. of SLT*, Palm Beach, Aruba.

E. Levin, R. Pieraccini, and W. Eckert. 2000. A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies. *IEEE Trans. on Speech and Audio Processing*, 8(1):11–23.

O. Pietquin. 2004. *A Framework for Unsupervised Learning of Dialogue Strategies*. Ph.D. thesis, Faculte Polytechnique de Mons.

V. Rieser and O. Lemon. 2006. Cluster-based User Simulations for Learning Dialogue Strategies. In *Proc. of ICSLP*, Pittsburgh, PA.

J. Schatzmann, K. Weilhammer, M.N. Stuttle, and S. Young. 2006. A Survey of Statistical User Simulation Techniques for Reinforcement-Learning of Dialogue Management Strategies. *KER*, 21(2):97–126.

J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young. 2007. Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Proc. of HLT/NAACL*. Rochester, NY.

K. Scheffler and S. Young. 2002. Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. In *Proc. of HLT*. San Diego, CA.

B. Thomson, J. Schatzmann, K. Weilhammer, H. Ye, , and S. Young. 2007. Training a real-world POMDP dialogue system. In *Proc. of HLT/NAACL Workshop: Bridging the Gap*. Rochester, NY.

X. Wei and A.I. Rudnicky. 1999. An agenda-based dialog management architecture for spoken language systems. In *Proc. of IEEE ASRU*. Seattle, WA.

J. D. Williams and S. Young. 2005. Scaling Up POMDPs for Dialog Management: The “Summary POMDP” Method. In *Proc. of ASRU*. San Juan, Puerto Rico.

S. Young, J. Williams, J. Schatzmann, M. Stuttle, and K. Weilhammer. 2005. The hidden information state approach to dialogue management. Technical Report CUED/F-INFENG/TR.544, Cambridge University.

S. Young, J. Schatzmann, K. Weilhammer, and H. Ye. 2007. The Hidden Information State Approach to Dialog Management. In *Proc. of ICASSP*, Honolulu, HI.

S. Young. 2002. Talking to machines (statistically speaking). In *Proc. of ICSLP*. Denver, CO.

7 Appendix

7.1 Sample dialogue and user state sequence

Initialisation *(Generate goal constraints and requests and populate the agenda)*

$$C_0 = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \end{bmatrix} \quad R_0 = \begin{bmatrix} name = \\ addr = \\ phone = \end{bmatrix} \quad A_0 = \begin{bmatrix} inform(type = bar) \\ inform(drinks = beer) \\ inform(area = central) \\ request(name) \\ request(addr) \\ request(phone) \\ bye() \end{bmatrix}$$

Sys 0 Hello, how may I help you? *(Push 0 items onto the agenda)*
Usr 1 I'm looking for a nice bar serving beer. *(Pop 2 items off the agenda)*

$$C'_1 = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \end{bmatrix} \quad R'_1 = \begin{bmatrix} name = \\ addr = \\ phone = \end{bmatrix} \quad A'_1 = \begin{bmatrix} inform(area = central) \\ request(name) \\ request(addr) \\ request(phone) \\ bye() \end{bmatrix}$$

Sys 1 Ok, a wine bar. What price range? *(Add 1 constraint, push 2 items onto the agenda)*

$$C_2 = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \\ prange = cheap \end{bmatrix} \quad R_2 = \begin{bmatrix} name = \\ addr = \\ phone = \end{bmatrix} \quad A_2 = \begin{bmatrix} negate(drinks = beer) \\ inform(prange = cheap) \\ inform(area = central) \\ request(name) \\ request(addr) \\ request(phone) \\ bye() \end{bmatrix}$$

Usr 2 No, beer please! *(Pop 1 item off the agenda)*
Sys 2 A bar serving beer, correct? *(Push 1 item onto the agenda)*

$$C_3 = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \\ prange = cheap \end{bmatrix} \quad R_3 = \begin{bmatrix} name = \\ addr = \\ phone = \end{bmatrix} \quad A_3 = \begin{bmatrix} affirm() \\ inform(prange = cheap) \\ inform(area = central) \\ request(name) \\ request(addr) \\ request(phone) \\ bye() \end{bmatrix}$$

Usr 3 Yeah something cheap in the town centre. *(Pop 3 items off the agenda)*

$$C'_3 = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \\ prange = cheap \end{bmatrix} \quad R'_3 = \begin{bmatrix} name = \\ addr = \\ phone = \end{bmatrix} \quad A'_3 = \begin{bmatrix} request(name) \\ request(addr) \\ request(phone) \\ bye() \end{bmatrix}$$

Sys 3 Murphy's on Main Sq. serves cheap beer. *(Fill 2 requests, delete agenda requests for filled slots)*

$$C_4 = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \\ prange = cheap \end{bmatrix} \quad R_4 = \begin{bmatrix} name = Murphy's \\ addr = Main\ Sq \\ phone = \end{bmatrix} \quad A_4 = \begin{bmatrix} request(phone) \\ bye() \end{bmatrix}$$

Usr 4 And what's the phone number? *(Pop 1 item off the agenda)*
Sys 4 The number is 796 69 94. *(Fill 1 request)*

$$C_5 = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \\ prange = cheap \end{bmatrix} \quad R_5 = \begin{bmatrix} name = Murphy's \\ addr = Main\ Sq \\ phone = 7966994 \end{bmatrix} \quad A_5 = \begin{bmatrix} bye() \end{bmatrix}$$

Usr 5 Thanks, goodbye! *(Pop 1 item off the agenda)*

Figure 5: Sample dialogue showing the state of the user goal and agenda. Note that system turn 1 *"What price range?"* triggers the user act *inform(prange=cheap)* to be pushed onto the agenda but it is not executed until turn 3 because *negate(drinks=beer)* is issued first.

7.2 Sample agenda update transition using the summary-space mapping

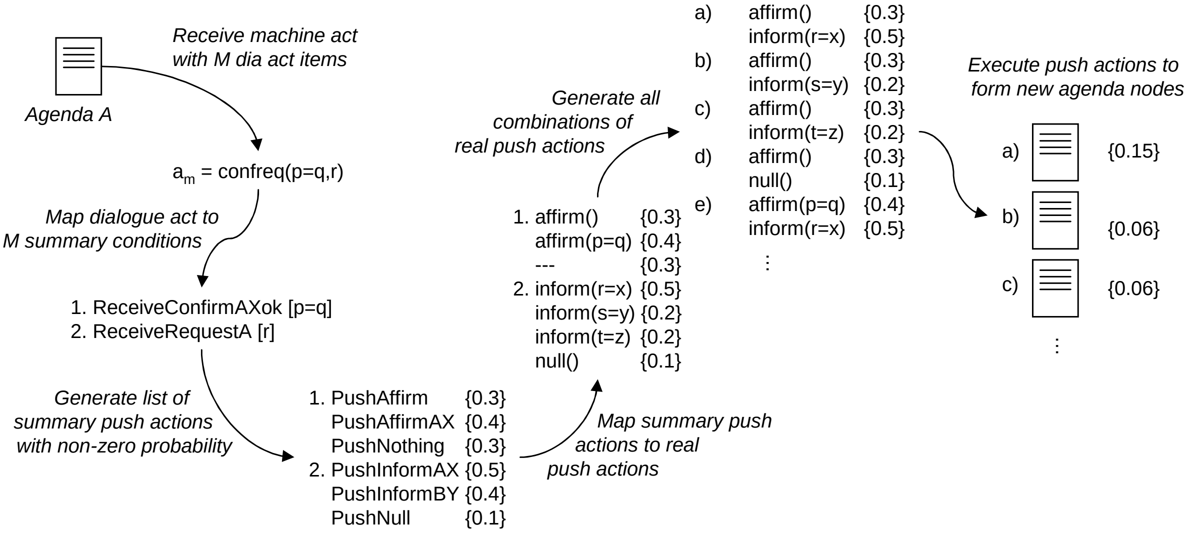

Figure 6: Simplified example illustrating the summary space technique for agenda updates.

The incoming machine act in this example is assumed to be a_m =*confreq(p=q,r)*, i.e. an implicit confirmation of the slot-value pair *p=q* and a request for the slot *r*. The update step proceeds as follows:

1. Based on the current state of the goal (not shown here), the first step is to map each dialogue act item (slot-value pair) to a summary condition z_{cond}. Given that the confirmation *p=q* in the example does not violate any of the constraints in the user goal, it is mapped to *ReceiveConfirmAXok[p=q]*. The request for *r* is mapped to *ReceiveRequestA[r]*.

2. A list of summary push actions z_{push}, each with probability $P(z_{push}|z_{cond})$, is now generated for each summary condition z_{cond}. A (shortened) list of examples is shown in the figure. The summary push action *PushInformAX*, for instance, implies that an *inform* act with the requested slot (in this case *r*) is pushed onto the agenda. Note that summary push actions with zero probability can be discarded at this point.

3. The summary push actions are now mapped to real push actions. This is a 1-to-1 mapping for most summary push actions, but some summary push actions can map to several real push actions. This is illustrated in the figure by the summary push action *PushInformBY*, which implies that the corresponding real push action is an *inform* dialogue act containing some slot-value pair *B=Y* other than the requested slot, in this case *s=y* or *t=z*. In such cases, the probability mass is split evenly between the real push actions for a summary push action, as shown in the figure.

4. Using one real push action from each summary condition, a list of all possible combinations of push actions is now generated. Each combination represents a series of dialogue acts to be pushed onto the agenda. As shown in the figure, each combination is used to create a new agenda. The transition probability is computed as the product of the real push actions that were used to make the transition.

Note that the set of summary conditions and summary actions is independent of the number of concepts and database entries, allowing the method to scale to more complex problem domains and larger databases.