USING PARTICLE FILTERS TO TRACK DIALOGUE STATE

Jason D. Williams

AT&T Labs – Research, Shannon Laboratory, 180 Park Ave., Florham Park, NJ 07932, USA

jdw@research.att.com

ABSTRACT

The benefit of tracking a *probability distribution* over multiple dialogue states has been demonstrated in the literature. However, the dialogue state in past work has been limited to a small number of variables, and growing the number of variables in the dialogue state prevents the probability distribution from being updated in real-time. This paper shows how the number of variables composing the dialogue state can be increased while maintaining response times suitable for a spoken dialogue system. Rather than performing exact inference using the joint distribution over all variables, a particle filter is employed to compute an approximate update. Dialogue states (particles) are sampled, weighted by their agreement with the speech recognition results, and marginalized to produce a new distribution over each variable. Results on a spoken dialogue system for troubleshooting show that a relatively small number of particles are required to achieve performance close to an exact update, enabling the dialogue system to run in real-time.

Index Terms— dialogue modelling, dialogue management, spoken dialogue systems, particle filter, Monte Carlo

1. INTRODUCTION

Traditional dialogue systems maintain a single hypothesis of the dialogue state such as a form or frame. Recently, methods of maintaining a *distribution* over dialogue states have been shown to yield better performance, including decision theoretic methods [1], M-Best lists [2], and partially observable Markov decision processes (POMDPs) [3]. The intuition is that a distribution over dialogue states directly models both the errors introduced in recognition and the variability of the user's responses, which allows it to consider multiple dialogue histories. This enables a dialogue manager to better cope with speech recognition errors and ultimately to choose conversational actions more effectively.

Past research has assumed the dialogue state contains essentially one persistent hidden variable – the user's goal, such as a flight itinerary – and that this variable is fixed throughout the dialogue [1, 3, 4]. This is a significant limitation because in general there may be many hidden persistent variables which change state throughout the dialogue. For example, in the troubleshooting domain, in which a dialogue system helps a user to troubleshoot a product such as a failed DSL connection, there are numerous persistent hidden variables, such as the power state of the DSL modem, whether the username has been entered correctly, whether there is a service outage, and whether the network cable is connected correctly. These variables are interrelated and continuously changing state throughout the dialogue – indeed, the goal of the dialogue system is to guide each of the variables into a working state. Updating the distribution over all of these constantly-changing variables is impossible in real-time, and the dialogue literature has not tackled this problem.

This paper proposes a new approach to updating a distribution over dialogue states which allows the number of variables to be scaled. The dialogue state is cast as an arbitrary Bayesian network and *approximate* updates are performed with a *particle filter*, which is a general-purpose technique for approximate inference in Bayesian networks [5]. Dialogue states (particles) are sampled, weighted by their agreement with the speech recognition result, and marginalized. As particles are added, the accuracy of the estimate improves at the expense of additional computation.

This paper is organized as follows: section 2 reviews the probability update task, and explains relevant past work; section 3 introduces our spoken dialogue system and illustrates why the probability update is problematic; section 4 introduces particle filters and shows how they can be applied to spoken dialogue systems; section 5 explains our evaluation and provides results; and section 6 briefly concludes.

2. BACKGROUND

In general, a dialogue system can be framed as a Bayesian network consisting of the tuple $(\mathbb{S}, \mathbb{O}, \mathbb{A}, \mathbb{T}, \mathbb{Z}, b_0)$. $\mathbb{A}$ represents the set of actions available to the dialogue system (such as asking a question or consulting a database) with $a \in \mathbb{A}$, and $\mathbb{O}$ represents the set of observations the system may make about its environment (such as output from the ASR and understanding process or a database result) with $o \in \mathbb{O}$. $\mathbb{S}$ represents the space of possible dialogue states with $s \in \mathbb{S}$, and in practice s is usually decomposed into a number of component variables $s = (s_1, \ldots, s_N)$ which track, for example, the user's goal, the user's (true, unobservable) action, and the di-

alogue history. $\mathbb{T}$ provides a model of how the dialogue state changes in response to system actions $P(s'|s,a)$, and $\mathbb{Z}$ provides a model of how the observations relate to the system state $P(o'|s',a)$.

A key property of spoken dialogue systems is that the observation o provides noisy and incomplete information about the state of the dialogue s, and the Bayesian network accounts for this by tracking a distribution over dialogue states $b(s)$ called a *belief state*, with initial belief b_0. At each time-step, b is updated by summing over all possible state transitions:

$$b'(s') = \eta \cdot P(o'|s',a) \sum_{s} P(s'|s,a)b(s). \quad (1)$$

Substituting in $s = (s_1, \ldots, s_N)$,

$$b'(s'_1, \ldots, s'_N) = \eta \cdot P(o'|s'_1, \ldots, s'_N, a) \cdot \sum_{s_1,\ldots,s_N} P(s'_1, \ldots, s'_N|s_1, \ldots, s_N, a)b(s_1, \ldots, s_N) \quad (2)$$

where η is a normalization constant [6]. The process of maintaining b at each time step is called *belief monitoring*.

The belief state b is used at run-time to select a system action using some policy $\pi : b \rightarrow a$. This policy can be produced using techniques such as POMDPs [3], decision-theory [1], or by hand crafting [2]. The method used is not important to this paper; the key point is that all probabilistic techniques rely on being able to compute $b(s)$ in real time, as the dialogue is progressing.

When the number of possible dialogue states $|\mathbb{S}|$ is small, the update in equation 2 is straightforward. However, for dialogue systems of a realistic size, the number of possible dialogue states is very large. Since the dialogue state s is typically decomposed into component variables $s = (s_1, s_2, ..., s_N)$ with $s_i \in \mathcal{S}_i$, the total number of dialogue states is $\prod_i |\mathcal{S}_i|$, which grows exponentially in the number of variables.

Past work has focused on the slot-filling domain and sought to grow the number of distinct values that a single variable – the user's goal – can take on. For example, one can assume that each slot s_i can be tracked independently [3], avoiding computing a joint distribution over $(s_1, s_2, ..., s_N)$. Alternatively, an M-Best list of dialogue states can be used to approximate a distribution over all states by tracking only the hypotheses suggested by the ASR N-Best list [2], and it has been shown how this M-Best list of user goals can be maintained exactly [4]. Crucially, past work has assumed that the user's goal is fixed: in other words, past work has shown how to perform belief monitoring when there is a single persistent variable which takes on a fixed value. In this work, we tackle the problem of belief monitoring when there are many variables, and when the variables are constantly changing state, as is the case in domains such as troubleshooting. The next section shows an example dialogue system in this domain and illustrates why belief monitoring here is difficult.

3. EXAMPLE SPOKEN DIALOGUE SYSTEM

A general-purpose model for POMDP-based dialogue systems for troubleshooting has been previously presented in [7]. This model has been used to build a spoken dialogue system which helps a user restore a failed DSL connection that models many, but not all, aspects of DSL troubleshooting.

In our system, the dialogue state s is decomposed into 19 components, and the observation o is decomposed into 2 components, listed in Table 1, with dependencies shown in Figure 1. The models of user behavior (nodes 12 and 20) are stochastic and are estimated from annotated conversations between users and DSL technicians. The models of the product behavior (remainder of the state nodes) were handcrafted based on interviews with DSL technicians, and most of these models are deterministic: for example, if the power to the DSL modem is off (node 13), then the power and network lights will both be off (nodes 17 and 18). Concept recognition errors were generated with $p = 0.30$, and confidence scores were drawn from an exponential distribution such that (at an equal error rate confidence threshold) about half of the concept errors could be identified. The reward function gives -1 for each action until the end of the dialogue when it gives +100 for correctly restoring service or -100 for failing to restore service.

A dialogue manager was created using an optimization process described in [8]. This technique takes as input a POMDP model (i.e., a Bayesian network and a reward function) and a "seed" finite-state-based dialogue controller, and produces an improved dialogue controller which maximizes the sum of rewards gained over the dialogue. Here, the input seed finite-state-based controller was created by hand, reflecting the agents' troubleshooting practices [7]. Space limitations prevent a full description of the improvement algorithm; the intuition is that the algorithm exploits the belief state of the Bayesian network at runtime to "rewire" the dialogue controller to achieve an improvement in total reward. Below, this hand-crafted controller will be used as a baseline to measure the gain in performance achieved by performing belief monitoring.

Before each system turn, the belief state must be updated as shown in equation 2. This is problematic in this dialogue system because applying equation 2 directly requires iterating over all approximately 40 million dialogue states, requiring between 13s and 48s, which is clearly too slow to run in real time. (The response time varies because, at certain points in the dialogue, a variable's value may be known with certainty and this can be used to speed up belief monitoring: for example, if a DSL modem responds to a ping, then there is definitely not a service outage.) Moreover, standard techniques for passing evidence through the network incrementally (such as Junction Trees [9]) are not of help here: evidence exists at both the root nodes and leaf nodes, and because there are multiple paths from root to leaf, it is not possible to isolate ("D-separate") different parts of the network. To perform ex-

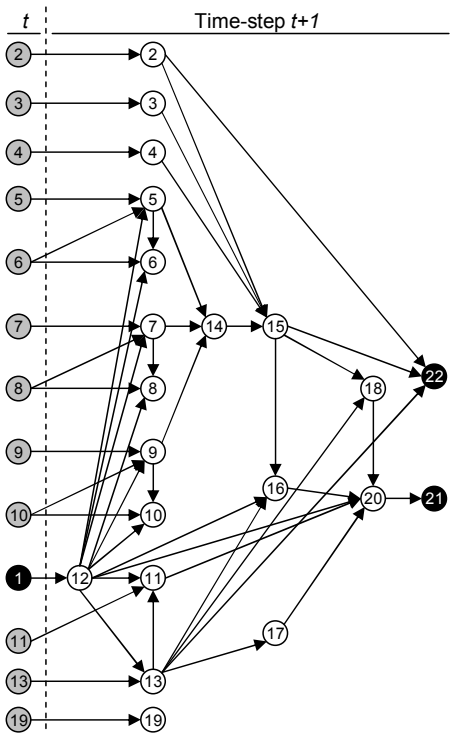

Fig. 1. Bayesian network showing the troubleshooting spoken dialogue system. Node labels refer to Table 1. Gray nodes have "soft" evidence (a distribution), black nodes have "hard" evidence (a known value), and the aim is to infer the posterior distribution over the white nodes.

Type	ID	Description	Size
A	1	System's action	19
S	2	Service outage	2
	3	Upstream network failure	2
	4	Unknown, unfixable problem	2
	5	Correct username in browser	2
	6	Correct username saved to modem	2
	7	Correct password in browser	2
	8	Correct password saved to modem	2
	9	Correct service type in browser	2
	10	Correct service type saved to modem	2
	11	Config screen visible in browser	2
	12	User's troubleshooting action	13
	13	State of DSL modem	3
	14	Modem configuration is correct	2
	15	DSL connection is working properly	2
	16	User successfully opened a webpage	2
	17	State of modem power light	2
	18	State of modem network light	3
	19	Dialogue is in progress vs. finished	2
	20	User's communicative action	11
O	21	ASR/NLU result, incl. conf. score	11
	22	Troubleshooting test result	2

Table 1. Description of the variables used in the troubleshooting dialogue system. A refers to action variables, S to state variables, and O to observation variables.

act inference, we are forced to compute the joint distribution. Since this is impossible in real-time, we instead turn to an approximation technique, described next.

4. PARTICLE FILTER METHOD

Particle filters are a general-purpose approximation technique for performing inference in Bayesian networks [5]. A particle filter operates by sampling values for unobservable variables, where each sample is called a "particle". Each particle is weighted by the likelihood that it would generate the observable evidence and these weights are normalized to produce an estimate of the posterior given the evidence.[1]

Here, a particle filter will be used to approximate the update in Equation 2. The method itself makes two approximations. First, the joint distribution $b(s_1, s_2, ..., s_N)$ is approximated by the product of the marginals:

$$b(s_1, s_2, ..., s_N) \approx \prod_i b_i(s_i) \tag{3}$$

[1]Gibbs sampling, another method for approximate inference, was also considered, but was found to be unsuitable because it cannot admit networks with deterministic variables.

so that the belief update can be restated as

$$b'(s'_1, \ldots, s'_N) \approx \eta \cdot P(o'|s'_1, \ldots, s'_N, a) \cdot \sum_{s_1,\ldots,s_N} P(s'_1, \ldots, s'_N|s_1, \ldots, s_N, a) \prod_i b_i(s_i). \tag{4}$$

This first assumption will enable particles to be sampled at each time-step. However, it also discards dependencies between the variables, and so empirical evaluation is important.

The particle filter itself is used to approximate Equation 4. The procedure is shown in Algorithm 1, which samples X particles for each update. For each particle, first values of variables in the current time-step are sampled by drawing a value $s_{(i)}$ from each marginal $b_i(s_i)$ (line 2). Then, values for variables in the next time-step $s'_{(j)}$ are sampled according to the transition dynamics (line 3) and saved as particle x (line 4). Finally, the weight for this particle is computed as the likelihood that it would have generated the observation (line 5). After sampling is complete, the particle weights are normalized (line 7), and the new estimated marginals are computed by summing the normalized weights for like variable values (line 8).

As the number of particles approaches infinity, the error (in the second approximation) goes to zero under mild assumptions [10]. The amount of computation and storage re-

Algorithm 1: Particle filtering process.

Input: $b_i(s_i), i = 1 \ldots N; a; o'$
Output: $b'_j(s'_j), j = 1 \ldots N$
1 **for** $x = 1$ **to** X **do**
2 $\quad s_{(i)} \sim b_i(s_i), \forall i$
3 $\quad s'_{(j)} \sim p(s'_j | s'_{(1)}, \ldots, s'_{(j-1)}, s_{(1)}, \ldots, s_{(N)}, a), \forall j$
4 $\quad p_x(j) = s'_{(j)}, \forall j$
5 $\quad w_x = p(o' | s'_{(1)}, \ldots, s'_{(N)}, a)$
6 **for** $x = 1$ **to** X **do**
7 $\quad \overline{w}_x = \frac{w_x}{\sum_{\hat{x}} w_{\hat{x}}}$
8 $b'_j(s'_j) = \sum_{x:p_x(j)=s'_j} \overline{w}_x, \forall j$

quired both grow linearly in the number of particles, so the number of particles sets the trade-off between speed and accuracy: as particles are added, the belief estimate improves at the expense of additional computation. This allows the method to be "tuned" to deliver a response within a specified time, which is an important property in a real-time environment such as a dialogue system.

While testing the method using small numbers of particles, it was noticed that occasionally observations which were always reliable (such as a network test operation) caused all particles to receive zero weight. This occurred when a speech recognition error earlier in the dialogue caused the belief in the correct (though unlikely) value for a variable to go unsampled and receive no probability mass. To prevent this from happening, the belief $b'_j(s'_j)$ (in line 8) is never allowed to go below a threshold. Experimentation found that a threshold of $1/(X * |\mathcal{S}_i|)$ works well in practice, and each variable value was always allocated at least this much mass. Since the reserved mass approaches zero as the number of particles approaches infinity, this modification does not change the asymptotic accuracy of the estimate.

5. EVALUATION

First, the accuracy of the method was evaluated. 500 simulated dialogues were produced using *exact* belief monitoring (i.e., enumerating the joint as in Equation 2). For each joint, the (true) marginal over each variable was then computed. Next, the sequence of system actions and observations generated in these dialogues were provided to the particle filter method, and its estimate of each variable's marginal at each time-step was obtained. For each time-step in each dialogue, the true and estimated marginals were compared for each variable, and the maximum L1 error across all variables was computed. These L1 errors were averaged across dialogues to obtain an average error per time-step.

Results are shown in Figure 2 for various numbers of particles. When only 10 particles are used, the error quickly rises to nearly its upper bound of 1.0, indicating a complete mis-

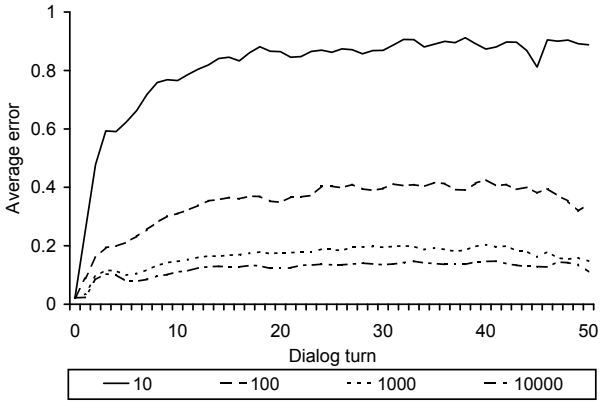

Fig. 2. Average error for various numbers of particles.

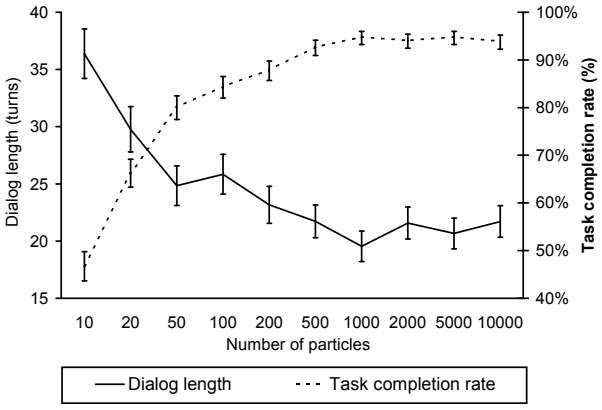

Fig. 3. Number of particles vs. average dialogue length and task completion rate. Error bars are 95% confidence interval.

estimate of the belief in at least one the variable values. As the number of particles is increased, the average error decreases. As has been found in other domains such as robotics [11], the error appears to plateau after a few time-steps rather than steadily growing. This is an important result because it indicates that the error is likely to remain constant over the course of the dialogue rather than steadily increasing, suggesting the method is suitable for both long and short dialogues.

In practice we are not interested in estimation error, but rather in the performance of the spoken dialogue system in terms of task completion rate and dialogue length. In other words, estimation error is only significant if it impacts performance. To measure this, simulated dialogues were run using approximate belief monitoring for various numbers of particles. Results are shown in Figure 3. As the number of particles is increased, task completion increases and dialogue length decreases, to an asymptote at about 1000 particles.

This performance was then compared to two baselines, previously presented in [7]. The first baseline uses exact (joint) belief monitoring, which shows performance if no approximations are made. This provides an upper bound on perfor-

	Particle Filter	Exact	Hand-crafted
N	1000	1000	1000
Reward	71.0	75.3	6.6
TCR	94.8%	96.1%	77.8%
Length	19.5	17.9	76.8
Response Time	3.5-3.8s	13-48s	0.91-1.4s

Table 2. Performance of particle filter (with 1000 particles), exact updating, and a hand-crafted dialogue controller. N indicates the number of simulated dialogues; TCR is task completion rate; and dialogue length is measured in turns.

mance for the particle filter method. The second baseline does not perform belief monitoring at all, but rather uses the same hand-crafted finite-state-based dialogue controller which was used as a "seed" for POMDP optimization in section 3. This second baseline provides an indication of the gain in performance resulting from the addition of belief monitoring. 1000 simulated dialogues were run using the particle filter method and each of the baselines. Results are shown in table 2. The particle filter method incurs a slight decrease in performance compared to exact belief monitoring, and maintains a significant margin of improvement over the hand-crafted dialogue manager.

Our ultimate aim is to obtain good performance in a real-time environment, and so we lastly investigated response time. Response time was measured by installing each dialogue manager in an end-to-end spoken dialogue system. Several dialogues were run with each of the two baselines, and with the particle filter method using various numbers of particles. In each turn of each dialogue, the time between the end of the user's speech and the beginning of the system response was measured. During this time, the dialogue system is performing speech recognition, the belief state update, action selection, and text-to-speech generation.

Results are shown in Figure 4 and also in Table 2. The end-to-end response time of the particle filter with 1000 (or fewer) particles is faster than exact belief monitoring. Moreover, the response time of the particle filter is less variable than exact updating: for example, the particle filter with 1000 particles ranges from 3.5s to 3.8s, whereas exact updating ranges from 13s to 48s. This illustrates another benefit of the particle filter method: whereas the running time of an exact update depends on the current belief state, the running time of the particle filter scales with the number of particles and is largely invariant to the current belief state. In other words, the running time of the particle filter method is more predictable, which is important in a real-time environment.

In sum, whereas exact belief monitoring is impossible in real-time, a particle filter yields an acceptable response time while incurring only a small decrease in performance. In addition, the particle filter out-performs a hand-crafted baseline by a very large margin while incurring an acceptable increase in response time.

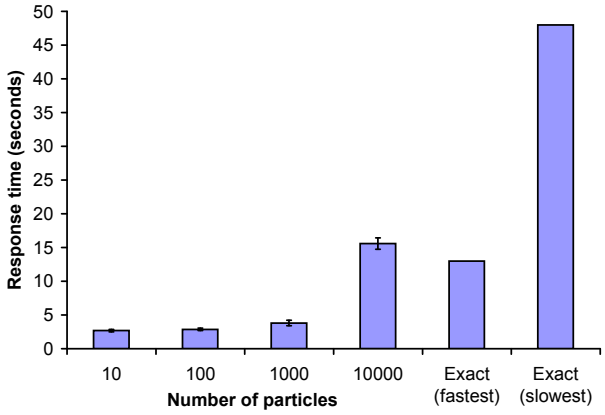

Fig. 4. Number of particles vs. response time. Error bars show 1 standard deviation.

6. CONCLUSIONS

This paper has shown how to use a particle filter for belief monitoring in probabilistic spoken dialogue systems. In the troubleshooting domain under dialogue simulation, the particle filter approximation achieved performance very close to exact calculation while providing significant computational savings, enabling belief monitoring to run in real-time.

Past work has addressed how to perform belief monitoring efficiently when there is a single persistent variable with fixed state; this paper has demonstrated how particle filters can be used to tackle belief monitoring when there are many variables with constantly changing state. Even so, in the evaluation presented here, the number of values for each variable was rather small, and it seems likely that the number of particles required may grow with the size of the largest variable. We intend to explore this in future work.

7. ACKNOWLEDGEMENTS

Thanks to Patrick Wolfe and Narendra Gupta for helpful discussions.

8. REFERENCES

[1] T Paek and E Horvitz, "Conversation as action under uncertainty," in *Proc Conf on Uncertainty in Artificial Intelligence (UAI), Stanford, California*, 2000.

[2] H Higashinaka, M Nakano, and K Aikawa, "Corpus-based discourse understanding in spoken dialogue systems," in *Proc Association for Computational Linguistics (ACL), Sapporo, Japan*, 2003.

[3] JD Williams, *Partially Observable Markov Decison Processes for Dialog Management*, Ph.D. thesis, Cambridge University, 2006.

[4] SJ Young, J Schatzmann, K Weilhammer, and H Ye, "The hidden information state approach to dialog management," in *Proc Intl Conf on Acoustics, Speech, and Signal Processing (ICASSP), Honolulu, Hawaii, USA*, 2007.

[5] N Pitt and N Shephard, "Filtering via simulation: auxiliary particle filter," *Journal of the American Statistical Association*, vol. 94, no. 446, 1999.

[6] L Kaelbling, ML Littman, and AR Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, 1998.

[7] JD Williams, "Applying POMDPs to dialog systems in the troubleshooting domain," in *NAACL-HLT Workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technologies, Rochester, New York, USA*, 2007.

[8] JD Williams, P Poupart, and SJ Young, "Partially observable Markov decision processes with continuous observations for dialogue management," in *Proc SIGdial Workshop on Discourse and Dialogue, Lisbon*, 2005.

[9] F Jensen, *Bayesian networks and decision graphs*, Springer Verlag, 2001.

[10] MA Tanner, *Tools for Statistical Inference*, SpringerVerlag, 1993.

[11] K Kanazawa, D Koller, and SJ Russell, "Stochastic simulation algorithms for dynamic probabilistic networks," in *Proc Conf on Uncertainty in Artificial Intelligence (UAI), Montreal, Quebec, Canada*, 1995.