Optimal Dialog in Consumer-Rating Systems using a POMDP Framework

Zhifei Li
Center for Language and Speech Processing
Johns Hopkins University
Baltimore, MD 21218, USA
zhifei.work@gmail.com

Patrick Nguyen, Geoffrey Zweig
Microsoft Corporation
1 Microsoft Way,
Redmond, WA 98052, USA
{panguyen,gzweig}@microsoft.com

Abstract

Voice-Rate is an experimental dialog system through which a user can call to get product information. In this paper, we describe an optimal dialog management algorithm for Voice-Rate. Our algorithm uses a POMDP framework, which is probabilistic and captures uncertainty in speech recognition and user knowledge. We propose a novel method to learn a user knowledge model from a review database. Simulation results show that the POMDP system performs significantly better than a deterministic baseline system in terms of both dialog failure rate and dialog interaction time. To the best of our knowledge, our work is the first to show that a POMDP can be successfully used for disambiguation in a complex voice search domain like Voice-Rate.

1 Introduction

In recent years, web-based shopping and rating systems have provided a valuable service to consumers by allowing them to shop products and share their assessments of products online. The use of these systems, however, requires access to a web interface, typically through a laptop or desktop computer, and this restricts their usefulness. While mobile phones also provide some web access, their small screens make them inconvenient to use. Therefore, there arises great interests in having a spoken dialog interface through which a user can call to get product information (e.g., price, rating, review, etc.) on the fly. Voice-Rate (Zweig et al., 2007) is such a system. Here is a typical scenario under which shows the usefulness of the Voice-Rate system. A user enters a store and finds that a digital camera he has not planned to buy is on sale. Before he decides to buy the camera, he takes out his cell phone and calls Voice-Rate to see whether the price is really a bargain and what other people have said about the camera. This helps him to make a wise decision. The Voice-Rate system (Zweig et al., 2007) involves many techniques, e.g., information retrieval, review summarization, speech recognition, speech synthesis, dialog management, etc. In this paper, we mainly focus on the dialog management component.

When a user calls Voice-Rate for the information of a specific product, the system needs to identify, from a database containing millions of products, the *exact* product the user intends. To achieve this, the system first solicits the user for the product name. Using the product name as a query, the system then retrieves from its database a list of products related to the query. Ideally, the highest-ranked product should be the one intended by the user. In reality, this is often not the case due to various reasons. For example, there might be a speech recognition error or an information retrieval ranking error. Moreover, the product name is usually very ambiguous in identifying an exact product. The product name that the user says may not be exactly the same as the name in the product database. For example, while the user says *"Canon Powershot SD750"*, the exact name in the product database may be *"Canon Powershot SD750 Digital Camera"*. Even the user says the *exact* name, it is possible that the same name may be corresponding to different products in different categories, for instance books and movies.

Due to the above reasons, whenever the Voice-Rate system finds multiple products matching the user's initial speech query, it initiates a dialog procedure to identify the intended product by asking questions about the products. In the product database,

Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue, pages 104–111,
Columbus, June 2008.

many attributes can be used to identify a product. For example, a digital camera has the product name, category, brand, resolution, zoom, etc. Given a list of products, different attributes may have different ability to distinguish the products. For example, if the products belong to many categories, the category attribute is very useful to distinguish the products. In contrast, if all the products belong to a single category, it makes no sense to ask a question on the category. In addition to the variability in distinguishing products, different attributes may require different knowledge from the user in order for them to answer questions about these attributes. For example, while most users can easily answer a question on *category*, they may not be able to answer a question on the *part number* of a product, though the *part number* is unique and perfect to distinguish products. Other variabilities are in the difficulty that the attributes impose on speech recognition and speech synthesis. Clearly, given a list of products and a set of attributes, what questions and in what order to ask is essential to make the dialog successful. Our goal is to *dynamically* find such important attributes at each stage/turn.

The baseline system (Zweig et al., 2007) asks questions only on product name and category. The order of questions is fixed: first ask questions on product category, and then on name. Moreover, it is deterministic and does not model uncertainly in speech recognition and user knowledge. Partially observable Markov decision process (POMDP) has been shown to be a general framework to capture the uncertainty in spoken dialog systems. In this paper, we present a POMDP-based probabilistic system, which utilizes rich product information and captures uncertainty in speech recognition and user knowledge. We propose a novel method to learn a user knowledge model from a review database. Our simulation results show that the POMDP-based system improves the baseline significantly.

To the best of our knowledge, our work is the first to show that a POMDP can be successfully used for disambiguation in a complex voice search domain like Voice-Rate.

2 Voice-Rate Dialog System Overview

Figure 1 shows the main flow in the Voice-Rate system with simplification. Specifically, when a user calls Voice-Rate for the information of a specific product, the system first solicits the user for the product name. Treating the user input as a *query* and the product names in the product database as *documents*, the system retrieves a list of products that match the user input based on TF-IDF measure. Then, the *dialog manager* dynamically generates questions to identify the specific intended product. Once the product is found, the system plays back its rating information. In this paper, we mainly focus on the *dialog manager* component.

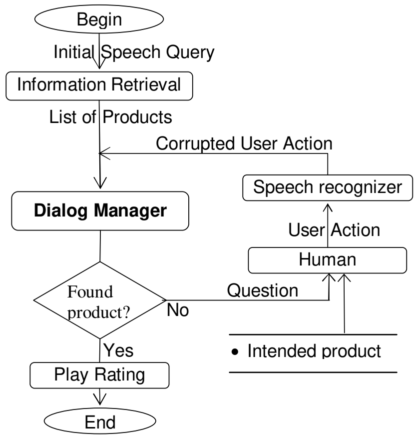

Figure 1: Flow Chart of Voice-Rate System

Step-1:	remove products that do not match the user action
Step-2:	any *category* question to ask? **yes:** ask the question and return **no:** go to step-3
Step-3:	ask a *product name* question

Table 1: Baseline Dialog Manager Algorithm

Baseline Dialog Manager: Table 1 shows the baseline dialog manager. In Step-1, it removes all the products that are not consistent with the user response. For example, if the user answers "camera" when given a question on *category*, the system removes all the products that do not belong to category "camera". In Step-2 and Step-3, the baseline system asks questions about product name and product category, and product category has a higher priority.

3 Overview of POMDP

3.1 Basic Definitions

A Partially Observable Markov Decision Process (POMDP) is a general framework to handle uncertainty in a spoken dialog system. Following notations in Williams and Young (2007), a POMDP is defined as a tuple $\{S, A, T, R, O, Z, \lambda, \vec{b}_0\}$ where S is a set of states s describing the environment; A is a set of machine actions a operating on the environment; T defines a transition probability $P(s^{'}|s, a)$; R defines a reward function $r(s, a)$; O is a set of observations o, and an observation can be thought as a corrupted version of a user action; Z defines an observation probability $P(o^{'}|s^{'}, a)$; λ is a geometric discount factor; and $\vec{b}_0$ is an initial belief vector.

The POMDP operates as follows. At each time-step (a.k.a. *stage*), the environment is in some unobserved state s. Since s is not known exactly, a distribution (called a *belief vector* $\vec{b}$) over possible states is maintained where $\vec{b}(s)$ indicates the probability of being in a particular state s. Based on the current belief vector $\vec{b}$, an *optimal action selection* algorithm selects a machine action a, receives a reward r, and the environment transits to a new unobserved state $s^{'}$. The environment then generates an observation $o^{'}$ (i.e., a user action), after which the system update the belief vector $\vec{b}$. We call the process of adjusting the belief vector $\vec{b}$ at each stage *"belief update"*.

3.2 Applying POMDP in Practice

As mentioned in Williams and Young (2007), it is not trivial to apply the POMDP framework to a specific application. To achieve this, one normally needs to design the following three components:

- State Diagram Modeling
- Belief Update
- Optimal Action Selection

The *state diagram* defines the topology of the graph, which contains three kinds of elements: *system state*, *machine action*, and *user action*. To drive the transitions, one also needs to define a set of models (e.g., user goal model, user action model, etc.). The modeling assumptions are application-dependent. The state diagram, together with the models, determines the dynamics of the system.

In general, the *belief update* depends on the observation probability and the transition probability, while the transition probability itself depends on the modeling assumptions the system makes. Thus, the exact belief update formula is application-specific.

Optimal action selection is essentially an optimization algorithm, which can be defined as,

$$a^* = \arg\max_{a \in A} G(P(a)), \quad (1)$$

where A refers to a set of machine actions a. Clearly, the optimal action selection requires three sub-components: a goodness measure function G, a prediction algorithm P, and a search algorithm (i.e., the *argmax* operator). The prediction algorithm is used to predict the behavior of the system in the future if a given machine action a was taken. The search algorithm can use an exhaustive linear search or an approximated greedy search depending on the size of A (Murphy, 2000; Spaan and Vlassis, 2005).

4 POMDP Framework in Voice-Rate

In this section, we present our instantiation of POMDP in the Voice-Rate system.

4.1 State Diagram Modeling

4.1.1 State Diagram Design

Table 2 summarizes the main design choices in the state diagram for our application, i.e., identifying the intended product from a large list of products.

As in Williams and Young (2007), we incorporate both the user goal (i.e., the intended product) and the user action in the *system state*. Moreover, to efficiently update belief vector and compute optimal action, the state space is dynamically generated and pruned. In particular, instead of listing all the possible combinations between the products and the user actions, at each stage, we only generate states containing the products and the user actions that are relevant to the last machine action. Moreover, at each stage, if the belief probability of a product is smaller than a threshold, we prune out this product and all its associated system states. Note that the intended product may be pruned away due to an overly large threshold. In the simulation, we will use a development set to tune this threshold.

As shown in Table 2, five kinds of *machine actions* are defined. The questions on product names are usually long, imposing difficulty in speech synthesis/recgonition and user input. Thus, short questions (e.g., questions on category or simple attributes) are preferable. This partly motivate us to exploit rich product information to help the dialog.

Seven kinds of *user actions* are defined as shown in Table 2. Among them, the user actions "others", "not related", and "not known" are special. Specifically, to limit the question length and to ensure the

Component	Design	Comments
System State	(Product, User action)	e.g., (HP Computer, Category: computer)
Machine Action	Question on *Category*	e.g., choose category: Electronics, Movie, Book
	Question on *Product name*	e.g., choose product name: Canon SD750 digital camera, Canon Powershot A40 digital camera, Canon SD950 digital camera, Others
	Question on *Attribute*	e.g., choose memory size: 64M, 128M, 256M
	Confirmation question	e.g., you want Canon SD750 camera, yes or no?
	Play Rating	e.g., I think you want Canon SD750 digital camera, here is the rating!
User Action	Category	e.g., Movie
	Product name	e.g., Canon SD750 digital camera
	Attribute value	e.g., memory size: 64M
	Others	used when a question has too many possible options
	Yes/No	used for a confirmation question
	Not related	used if the intended product is unrelated to the question
	Not known	used if the user does not have required knowledge to answer the question

Table 2: State Diagram Design in Voice-Rate

human is able to memorize all the options, we restrict the number of options in a single question to a threshold N (e.g., 5). Clearly, given a list of products and a question, there might be more than N possible options. In such a case, we need to merge some options into the "others" class. The third example in Table 2 shows an example with the "others" option. One may exploit a clustering algorithm (e.g., an iterative greedy search algorithm) to find an optimal merge. In our system, we simply take the top-(N-1) options (ranked by the belief probabilities) and treat all the remaining options as "others".

The "not related" option is required when some candidate products are irrelevant to the question. For example, when the system asks a question regarding the attribute "cpu speed" while the products contain both books and computers, the "not related" option is required in case the intended product is a book.

Lastly, while some attributes are very useful to distinguish the products, a user may not have enough knowledge to answer a question on these attributes. For example, while there is a unique *part number* for each product, however, the user may not know the exact part number for the intended product. Thus, "not known" option is required whenever the system expects the user is unable to answer the question.

4.1.2 Models

We assume that the user does not change his goal (i.e., the intended product) along the dialog. We also assume that the user *rationally* answers the question to achieve his goal. Additionally, we assume that the speech synthesis is good enough such that the user always gets the right information that the system intends to convey. The two main models that we consider include an observation model that captures speech recognition uncertainty, and a user knowledge model that captures the variability of user knowledge required for answering questions on different attributes.

Observation Model: Since the speech recognition engine we are using returns only a one-best and its confidence value $C \in [0, 1]$. We define the observation function as follows,

$$P(\widehat{a}_u|a_u) = \begin{cases} C & \text{if } \widehat{a}_u = a_u, \\ \frac{1-C}{|A_u|-1} & \text{otherwise.} \end{cases} \quad (2)$$

where a_u is the true user action, $\widehat{a}_u$ is the speech recognition output (i.e., corrupted user action), and A_u is the set of user actions related to the last machine action.

User Knowledge Model: In most of the applications (Roy et al., 2000; Williams, 2007) where the POMDP framework got applied, it is normally assumed that the user needs only common sense to answer the questions asked by the dialog system. Our application is more complex as the product information is very rich. A user may have different difficulty in answering different questions. For example, while a user can easily answer a question on *category*, he may not be able to answer a question on the *part number*. Thus, we define a user knowledge model to capture such uncertainty. Specifically, given a question (say a_m) and an intended product (say g_u) in the user's mind, we want to know how likely the user has required knowledge to answer the question. Formally, the user knowledge model is,

$$P(a_u|g_u, a_m) = \begin{cases} P(\textit{unk}|g_u, a_m) & \text{if } a_u\text{=}\textit{unk}, \\ 1 - P(\textit{unk}|g_u, a_m) & \text{if } a_u\text{=}\textit{truth}, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

where *unk* represents the user action "not known". Clearly, given a specific product g_u and a specific question a_m, there is exactly one correct user action (represented by *truth* in Equation 3), and its probability is $1 - P(\textit{unk}|g_u, a_m)$. Now, to obtain a user knowledge model, we only need to obtain $P(\textit{unk}|g_u, a_m)$. As shown in Table 2, there are four kinds of *question-type* machine actions a_m. We assume that the user always has knowledge to answer a question regarding the category and product name, and thus $P(\textit{unk}|g_u, a_m)$ for these types of machine actions are zero regardless of what the specific product g_u is. Therefore, we only need to consider $P(\textit{unk}|g_u, a_m)$ when a_m is a question about an attribute (say *attr*). Moreover, since there are millions of products, to deal with the data sparsity issue, we assume $P(\textit{unk}|g_u, a_m)$ does not depends on a specific product g_u, instead it depends on only the category (say *cat*) of the product g_u. Therefore,

$$P(\textit{unk}|g_u, a_m) \approx P(\textit{unk}|\textit{cat,attr}). \quad (4)$$

Now, we only need to get the probability $P(\textit{unk}|\textit{cat,attr})$ for each attribute *attr* in each category *cat*. To learn $P(\textit{unk}|\textit{cat,attr})$, one may collect data from human, which is very expensive. Instead, we learn this model from a database of online reviews for the products. Our method is based on the following intuition: *if a user cares/knows about an attribute of a product, he will mention either the **attribute name**, or the **attribute value**, or **both** in his review of this product*. With this intuition, the occurrence frequency of a given *attr* in a given category *cat* is collected from the review database, followed by proper weighting, scaling and normalization, and thus $P(\textit{unk}|\textit{cat,attr})$ is obtained.

4.2 Belief Update

Based on the model assumptions in Section 4.1.2, the belief update formula for the state $(g_u, a_u^{'})$ is,

$$\vec{b}(g_u, a_u^{'}) = \quad (5)$$
$$k \times P(\widehat{a}_u^{'}|a_u^{'})P(a_u^{'}|g_u, a_m) \sum_{a_u \in A(g_u)} \vec{b}(g_u, a_u)$$

where k is a normalization constant. The $P(\widehat{a}_u^{'}|a_u^{'})$ is the observation function as defined in Equation 2, while $P(a_u^{'}|g_u, a_m)$ is the user knowledge model as defined in Equation 3. The $A(g_u)$ represents the set of user actions a_u related to the system states for which the intended product is g_u.

In our state representation, a single product g_u is associated with several states which differ in the user action a_u, and the belief probability of g_u is the sum of the probabilities of these states. Therefore, even there is a speech recognition error or an unintentional user mistake, the true product still gets a non-zero belief probability (though the true/ideal user action a_u gets a zero probability). Moreover, the probability of the true product will get promoted through later iterations. Therefore, our system has error-handling capability, which is one of the major advantages over the deterministic baseline system.

4.3 Optimal Action Selection

As mentioned in Section 3.2, the optimal action selection involves three sub-components: a prediction algorithm, a goodness measure, and a search algorithm. Ideally, in our application, we should minimize the time required to successfully identify the intended product. Clearly, this is too difficult as it needs to predict the infinite future and needs to encode the time into a reward function. Therefore, for simplicity, we predict only one-step forward, and use the entropy as a goodness measure[1]. Formally,

[1]Due to this approximation, one may argue that our model is more like the greedy information theoretic model in Paek and Chickering (2005), instead of a POMDP model. However, we believe that our model follows the POMDP modeling framework in general, though it does not involve reinforcement learning currently.

the optimization function is as follows:

$$a^* = \arg\min_{a \in A} H(\text{Products} \mid a), \quad (6)$$

where $H(\text{Products} \mid a)$ is the entropy over the belief probabilities of the products if the machine action a was taken. When predicting the belief vector using Equation 5, we consider only the user knowledge model and ignore the observation function[2].

In the above, we consider only the *question-type* machine actions. We also need to decide when to take the *play rating* action such that the dialog will terminate. Specifically, we take the *play rating* action whenever the belief probability of the most probable product is greater than a threshold. Moreover, the threshold should depend on the number of surviving products. For example, if there are fifty surviving products and the most probable product has a belief probability greater than 0.3, it is reasonable to take the *play rating* action. This is not true if there are only four surviving products. Also note that if we set the thresholds to too small values, the system may play the rating for a *wrong* product. We will use a development set to tune these thresholds.

4.3.1 Machine Action Filtering during Search

We use an exhaustive linear search for the operator *argmin* in Equation 6. However, additional filtering during the search is required.

Repeated Question: Since the speech response from the user to a question is *probabilistic*, it is quite possible that the system will choose the same question that has been asked in previous stages[3]. Since our product information is very rich, many different questions have the similar capability to reduce entropy. Therefore, during the search, we simply ignore all the questions asked in previous stages.

"Not Related" Option: While reducing entropy helps to reduce the confusion at the *machine* side, it does not measure the "weirdness" of a question to the *human*. For example, when the intended product is a book and the candidate products contain both books and computers, it is quite possible that the optimal action, based solely on entropy reduction, is a question on the attribute "cpu speed". Clearly, such a question is very weird to the human as he is looking for a book that has nothing related to "cpu speed". Though the user may be able to choose the "not related" option correctly after thinking for a while, it degrades the dialog quality. Therefore, for a given question, whenever the system predicts that the user will have to choose the "not related" option with a probability greater than a threshold, we simply ignore such questions in the search. Clearly, if we set the threshold as zero, we essentially eliminates the "not related" option. That is, at each stage, we generate questions only on attributes that apply to all the candidate products. Since we dynamically remove products whose probability is smaller than a threshold at each stage, the valid question set dynamically expands. Specifically, at the beginning, only very general questions (e.g., questions on category) are valid, then more refined questions become valid (e.g., questions on product brand), and finally very specific questions are valid (e.g, questions on product model). This leads to very natural behavior in identifying a product, i.e., **coarse to fine**[4]. It also makes the system adapt to the user knowledge. Specifically, as the user demonstrates deeper knowledge of the products by answering the questions correctly, it makes sense to ask more refined questions about the products.

5 Simulation Results

To evaluate system performance, ideally one should ask people to call the system, and manually collect the performance data. This is very expensive. Alternatively, we develop a simulation method, which is automatic and thus allow fast evaluation of the system during development[5]. In fact, many design choices in Section 4 are inspired by the simulation.

5.1 Simulation Model

Figure 2 illustrates the general framework for the simulation. The process is very similar to that in Figure 1 except that the human user and the speech

[2]Note that we ignore the observation function only in the prediction, not in real belief update.

[3]In a regular decision tree, the answer to a question is *deterministic*. It never asks the same question as that does not lead to any additional reduction of entropy. This problem is also due to the fact we do not have an explicit reward function.

[4]While the baseline dialog manager achieves the similar behavior by *manually* enforcing the order of questions, the system here *automatically* discovers the order of questions and the question set is much more richer than that in the baseline.

[5]However, we agree that simulation is not without its limitations and the results may not precisely reflect real scenarios.

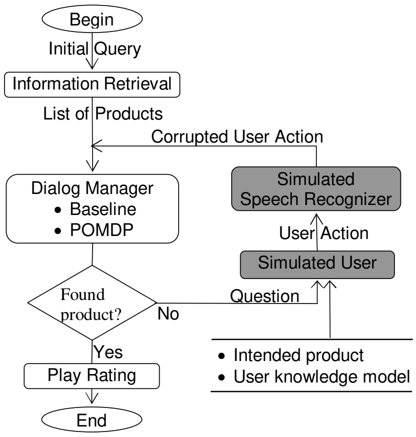

Figure 2: Flow Chart in Simulation

recognizer are replaced with a simulated component, and that the simulated user has access to a user knowledge model. In particular, we generate the user action and its corrupted version using random number generators by following the models defined in Equations 3 and 2, respectively. We use a fixed value (e.g., 0.9) for C in Equation 2.

Clearly, our goal here is not to evaluate the goodness of the user knowledge model or the speech recognizer. Instead, we want to see how the probabilistic dialog manger (i.e., POMDP) performs compared with the deterministic baseline dialog manager, and to see whether the richer attribute information helps to reduce the dialog interaction time.

5.2 Data Resources

In the system, we use three data resources: a product database, a review database, and a query-click database. The product database contains detailed information for 0.2 million electronics and computer related products. The review database is used for learning the user knowledge model. The query-click database contains 2289 pairs in the format (text query, product clicked). One example pair is (Canon Powershot A700, Canon Powershot A700 6.2MP digital camera). We divide it into a development set (1308 pairs) and a test set (981 pairs).

5.3 Results on Information Retrieval

For each initial query, the information retrieval (IR) engine returns a list of top-ranked products. Whether the intended product is in the returned list depends on the size of the list. If the intended product is in the list, the IR successfully recalled the product. Table 3 shows the correlation between the recall rate and the size of the returned list. Clearly, the larger the list size is, the larger the recall rate is. One may notice that the IR recall rate is low. This is because the query-click data set is very noisy, that is, the clicked product may be nothing to do with the query. For example, (msn shopping, Handspring Treo 270) is one of the pairs in our data set.

List Size	Recall Rate (%)
50	38.36
100	41.46
150	43.5

Table 3: Information Retrieval Recall Rates on Test set

5.4 Dialog System Configuration and Tuning

As mentioned in Section 4, several parameters in the system are configurable and tunable. Specifically, we set the max number of options in a question as 5, and the threshold for "not related" option as zero. We use the development set to tune the following parameters: the threshold of the belief probability below which the product is pruned, and the thresholds above which the most probable product is played. The parameters are tuned in a way such that no dialog error is made on the development set.

5.5 Results on Error Handling

Even the IR succeeds, the dialog system may not find the intended product successfully. In particular, the baseline system does not have error handling capability. Whenever the system makes a speech recognition error or the user mistakenly answers a question, the dialog system fails (either plays the rating for a wrong product or fails to find any product). On the contrary, our POMDP framework has error handling functionality due to its probabilistic nature. Table 5 compares the dialog error rate between the baseline and the POMDP systems. Clearly, the POMDP system performs much better to handle errors. Note that the POMDP system does not eliminate dialog failures on the test set because the **thresholds** are not perfect for the *test set*[6]. This is due to two reasons: the system may prune the intended product (*reason-1*), and the system may play the rating for a wrong product (*reason-2*).

[6]Note that the POMDP system does not have dialog failures on the *development set* as we tune the system in this way.

System	Size	Average			Max		
		Stages	Characters	Words	Stages	Characters	Words
Baseline	50	2.44	524.0	82.3	11	2927	546
	100	3.37	765.4	120.4	25	7762	1369
	150	3.90	906.4	143.0	30	9345	1668
POMDP	50	1.57	342.8	54.3	4	2659	466
	100	2.36	487.9	76.6	18	3575	597
	150	2.59	541.3	85.0	19	4898	767

Table 4: Interaction Time Results on Test Set

Size	Baseline (%)	POMDP (%)		
		Total	Reason-1	Reason-2
50	13.8	8.2	4.2	4.0
100	17.7	2.7	1.2	1.5
150	19.3	4.7	0.7	4.0

Table 5: Dialog Failure Rate on Test Set

5.6 Results on Interaction Time

It is quite difficult to measure the exact interaction time, so instead we measure it through the number of stages/characters/words required during the dialog process. Clearly, the number of characters is the one that matches most closely to the true time. Table 4 reports the average and maximum numbers. In general, the POMDP system performs much better than the baseline system. One may notice the difference in the number of stages between the baseline and the POMDP systems is not as significant as in the number of characters. This is because the POMDP system is able to exploit very short questions while the baseline system mainly uses the product name question, which is normally very long. The long question on product name also imposes difficulty in speech synthesis, user input, and speech recognition, though this is not reflected in the simulation.

6 Conclusions

In this paper, we have applied the POMDP framework into Voice-Rate, a system through which a user can call to get product information (e.g., price, rating, review, etc.). We have proposed a novel method to learn a user knowledge model from a review database. Compared with a deterministic baseline system (Zweig et al., 2007), the POMDP system is probabilistic and is able to handle speech recognition errors and user mistakes, in which case the deterministic baseline system is doomed to fail. Moreover, the POMDP system exploits richer product information to reduce the interaction time required to complete a dialog. We have developed a simulation model, and shown that the POMDP system improves the baseline system significantly in terms of both dialog failure rate and dialog interaction time. We also implement our POMDP system into a speech demo and plan to carry out tests through humans.

Acknowledgement

This work was conducted during the first author's internship at Microsoft Research; thanks to Dan Bohus, Ghinwa Choueiter, Yun-Cheng Ju, Xiao Li, Milind Mahajan, Tim Paek, Yeyi Wang, and Dong Yu for helpful discussions.

References

K. Murphy. 2000. A survey of POMDP solution techniques. Technical Report, U. C. Berkeley.

T. Paek and D. Chickering. 2005. The Markov assumption in spoken dialogue management. *In Proc of SIGdial 2005.*

N. Roy, J. Pineau, and S. Thrun. 2000. Spoken dialog management for robots. *In Proc of ACL 2000.*

M. Spaan and N. Vlassis. 2005. Perseus: randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195-220.

J. Williams. 2007. Applying POMDPs to Dialog Systems in the Troubleshooting Domain. *In Proc HLT/NAACL Workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technology.*

J. Williams and S. Young. 2007. Partially Observable Markov Decision Processes for Spoken Dialog Systems. *Computer Speech and Language* 21(2): 231-422.

G. Zweig, P. Nguyen, Y.C. Ju, Y.Y. Wang, D. Yu, and A. Acero. 2007. The Voice-Rate Dialog System for Consumer Ratings. *In Proc of Interspeech 2007.*