Integrating expert knowledge into POMDP optimization for spoken dialog systems

Jason D. Williams
AT&T Labs – Research, Shannon Laboratory, 180 Park Ave., Florham Park, NJ 07932, USA
jdw@research.att.com

Abstract

A common problem for real-world POMDP applications is how to incorporate expert knowledge and constraints such as business rules into the optimization process. This paper describes a simple approach created in the course of developing a spoken dialog system. A POMDP and conventional hand-crafted dialog controller run in parallel; the conventional dialog controller nominates a set of one or more actions, and the POMDP chooses the optimal action. This allows designers to express real-world constraints in a familiar manner, and also prunes the search space of policies. The method naturally admits compression, and the POMDP value function can draw on features from both the POMDP belief state and the hand-crafted dialog controller. The method has been used to build a full-scale dialog system which is currently running at AT&T Labs. An evaluation shows that this unified architecture yields better performance than using a conventional dialog manager alone, and also demonstrates an improvement in optimization speed and reliability vs. a pure POMDP.

Introduction

In many real-world commercial applications, control strategies are traditionally designed by hand. For complex systems, hand-design is often sub-optimal and partially observable Markov decision processes (POMDPs) provide a principled method for improving performance. However, commercial applications often require that any control strategy conform to a set of business rules, but there is no obvious way to incorporate these because in the classical POMDP formulation, the optimization process is free to choose any action at any time. In addition, system designers can readily articulate domain knowledge such as which actions must precede others, yet there is no straightforward way to communicate this to an optimization procedure. As a result, we desire a method for ensuring that a policy conforms to a set of constraints provided by a domain expert.

Spoken dialog systems provide a useful case study. Currently, in industry, a developer creates a detailed dialog plan by hand, encoding their knowledge of the task and business rules. For example, a system can be designed so that passwords are always verified before account access is granted. In addition, the dialog designer can craft each prompt to the current dialog context, and this is important because prompt wording has a strong effect on the user's speech patterns and satisfaction. This conventional approach is well-documented and has been used to develop hundreds of successful commercial dialog systems.

Even so, speech recognition technology remains imperfect: speech recognition errors are common and undermine dialog systems. To tackle this, the research community has begun applying POMDPs to dialog control (Roy, Pineau, and Thrun 2000; Zhang et al. 2001; Williams and Young 2007a; Young et al. 2007; Doshi and Roy 2007). POMDPs maintain a distribution over many hypotheses for the correct dialog state and choose actions using an optimization process, in which a developer specifies high-level goals via a reward function. In research settings, POMDP-based dialog systems have shown more robustness to speech recognition errors, yielding shorter dialogs with higher task completion rates.

However, commercial applications require that spoken dialog systems can be guaranteed to follow business rules. For example, it is obvious that the system should never print a ticket before it has asked for the origin city, but there is no direct way to communicate this to the optimization process, and it is unclear how to ensure that this will never happen in deployment. In addition, some policy decisions are obvious to designers—for example, confirmation should follow rather than precede questions—yet there is no easy way for a designer to ensure this is always the case. These issues are important hindrances to deploying POMDPs to real-world spoken dialog systems.

One simple way of encoding domain expertise is to adjust the POMDP reward function. However, this is not ideal because this complicates the state space design. It is also an unfamiliar method of expression for designers, and side-steps the real aim of imposing *constraints* rather than merely preferences. In the literature, expert knowledge has also been incorporated via active learning (Doshi, Pineau, and Roy 2008). While active learning seems attractive for accelerating optimization and model learning, it does not impose

hard constraints on the resulting policy.

In this paper, we describe a simple method for combining classical POMDPs with hand-crafted control strategies. A hand-crafted dialog manager and POMDP run in parallel, but the dialog manager is augmented so that it outputs one *or more* allowed actions at each time-step. The POMDP then choses the optimal action from this limited set. This has the effect of both requiring policies to conform to business rules, and enabling designers to express domain properties in a familiar manner. Further, because the set of policies considered by the optimization is informed by the developer, spurious action choices are pruned, and optimization runs faster and more reliably than in a classical POMDP.

We illustrate the technique on a real voice dialer application and show that the technique marries well with existing compression techniques and operates at scale. The remainder of this paper reviews the conventional and POMDP approaches, explains the method, presents the voice dialer application and illustrates application of the method, and provides results of a comparison with conventional techniques and a classical POMDP. An appendix details the policy estimation procedure.

Background

We begin by introducing the spoken dialog system domain and formalizing the problem. At each turn in a dialog, the dialog system takes a speech action a, such as "Where are you leaving from?". A user then responds with action u, such as "Boston". This u is processed by the speech recognition engine to produce an observation o, such as "AUSTIN". The dialog system examines o, updates its internal state, and outputs another a. The conventional (hand-design) and POMDP approaches differ in how they maintain this internal state, and how they choose actions given the state.

A conventional dialog controller can be viewed as a (possibly very large) finite state machine. This controller maintains a dialog state n (such as a form or frame) and relies on two functions for control, G and F. For a given state n, $G(n) = a$ decides which system action to output, and then after observation o has been received, $F(n, o) = n'$ decides how to update the state n to yield n'. This process repeats until the dialog is over. The important point is that G and F are written by hand, expressed programmatically in a language such as VoiceXML.

In the POMDP, there are a set of *hidden* states, where each hidden state s represents a possible state of the conversation, including quantities such as the user's action u, the user's underlying goals, and the dialog history (Williams and Young 2007a). Because the true state of the conversation isn't known, the POMDP maintains a *belief state* (probability distribution) over these hidden states, b, where $b(s)$ is the *belief* (probability) that s is the true state. By adding models of how the hidden state changes (transition function) and how the observation is corrupted (observation function), this distribution can be updated – i.e., $b'(s') = p(s'|a, o, b)$ – and there are methods for doing this efficiently and/or approximately (Young et al. 2006; Williams 2007). The belief state has the desirable property of accumulating information across all of the actions and observations over the course of the entire dialog history, and provides robustness to speech recognition errors.

For control, a developer specifies high-level goals in the form of a reward function, $R(s, a)$. R assigns a measure of goodness to each state/action pair and communicates, for example, the relative values of short dialogs and successful task completion. An optimization procedure then searches for the best action to take in each belief state in order to maximize the sum of rewards over the whole dialog. The result is a value function $Q(b, a)$, which estimates the long-term reward of taking action a at belief state b. The optimal action in belief state b is then $a^* = \arg\max_a Q(b, a)$.

In practice, the domain of $Q(b, a)$ is too large and compression is applied. In the spoken dialog system domain, the so-called "summary" method has achieved good results (Williams and Young 2007b). The intuition is to "summarize" b into a lower-dimensional feature vector $\hat{b}$, to summarize a into an *action mnemonic* $\hat{a}$, and to estimate a value function $\hat{Q}(\hat{b}, \hat{a})$ in this compressed space. For example, b might be a distribution over all cities and $\hat{b}$ might be the probability of the most likely city, $\hat{b} = \max_s b(s)$. Similarly, a might include actions to confirm any city (*confirm(london)*,*confirm(boston)*,etc.) whereas an action mnemonic $\hat{a}$ might compress this set to a single action like *confirm(most-likely-city)*.

Method

To unify these two approaches, several changes are made. The conventional dialog controller is extended in three respects: first, its action selection function $G(n) = a$ is changed to output a *set* of one *or more* allowable actions given a dialog state n, each with a corresponding summary action, $G(n) = \{(a_{(1)}, \hat{a}_{(1)}), \ldots, (a_{(M)}, \hat{a}_{(M)})\}$. Next, its transition function $F(n, o) = n'$ is extended to allow for different transitions depending on which of these actions was taken, and it is also given access to the resulting POMDP belief state, $F(n, a, o, b') = n'$. A (human) dialog designer still designs the contents of the state n and writes the functions G and F.

For action selection, compression will be applied but the state features used for action selection will be a function of both the belief state b *and* the dialog state n. This state feature vector is written $\hat{x}$ and is computed by a feature-function $H(b, n) = \hat{x}$. Note that these features may include both discrete elements and continuous elements. The POMDP value function is correspondingly re-cast to assign

values to these feature vectors, $\hat{Q}(\hat{x}, \hat{a})$.

The unified dialog manager operates as follows. At each time-step, the dialog manager is in state n and the POMDP is in belief state b. The dialog manager *nominates* a set of m allowable actions, where each action $a_{(m)}$ includes its summarized counterpart $\hat{a}_{(m)}$. The state features are computed as $\hat{x} = H(b, n)$. Then, the POMDP value function $\hat{Q}(\hat{x}, \hat{a})$ is evaluated for *only* those actions nominated by the dialog manager (not all actions), and the index m^* of the action that maximizes the POMDP value function is returned:

$$m^* = \arg \max_{m \in [1,M]} \hat{Q}(\hat{x}, \hat{a}_{(m)}).$$

Action $a_{(m^*)}$ is then output and reward r and observation o are received. The POMDP updates its belief state $b'(s') = p(s'|a_{(m^*)}, o, b)$ and the dialog manager transitions to dialog state $n' = F(n, a_{(m^*)}, o, b')$. An example of this process taken from the real system described below is shown in Figure 3.

Intuitively, the effect of constraining which actions are available prunes the space of policies, so if the constraints are well-informed, then optimization ought to converge to the optimal policy faster. Strictly speaking, the number of possible policies is still doubly exponential in the planning horizon. Nonetheless, in the results below, an reduction in planning complexity is observed.

In this method, action selection can be viewed as a general reinforcement learning problem, where states are feature vectors $\hat{x}$. This enables any general-purpose reinforcement learning technique to be applied which produces an estimate of $\hat{Q}(\hat{x}, \hat{a})$. However, because arbitrary compression may be used, the system dynamics may no longer be Markovian. In addition, the set of allowed actions is not a function of the feature vector $\hat{x}$, and so $\hat{Q}$ may mis-estimate future rewards. Because of these potential issues, it is important to verify the method on a real-world application, described next.

Example dialog system

We developed this method in the course of improving a voice dialer application. A hand-designed version of this voice dialer application has been accessible within the AT&T research lab for several years and receives daily calls. The dialer's vocabulary consists of the names of 50,000 AT&T employees. Since many employees have the same name, the dialer can disambiguate by asking for the callee's location. The dialer can also disambiguate between multiple phone listings for the same person (office and mobile) and can indicate when a callee has no number listed. This hand-designed dialog manager tracks a variety of elements in its state n, including the most recently recognized callee, how many callees share that callee's name, whether the callee has been confirmed, and many others. This existing dialog controller was used as our baseline, labelled as "HC" in the results. Our goal was to improve task completion and reduce dialog length by improving the action selection process (retaining the existing scope of functionality and set of actions).

The POMDP was then created. The belief state followed the SDS-POMDP model (Williams and Young 2007a) and maintained a belief state over all callees. The user model and speech recognition models used to update the belief state were based on transcribed logs from 320 calls.

The existing dialer was then extended in two respects. First, rather than tracking the most recently recognized callee, it instead obtained the most likely callee from the POMDP belief state. Second, it was altered to nominate a set of one or more allowable actions using knowledge about this domain. For example, on the first turn of the dialog, the only allowed action was to ask for the callee's name. Once a callee has been recognized, the callee can be queried again or confirmed. Additional actions are allowed depending on the properties of the most likely callee – for example, if the top callee is ambiguous, then asking for the callee's city and state is allowed; and if the top callee has both a cellphone and office phone listed, then asking for the type of phone is allowed. The transfer action is permitted only after the system has attempted confirmation. This unified controller was called "HC+POMDP".

For comparison, another controller was created which nominated every action at every time-step. Its actions also acted on the most likely callee in the belief state but no other restrictions were imposed. It could, for example, transfer a call to a callee who has not been confirmed, or ask for the city and state even if the top callee was not ambiguous. This controller was called "POMDP".

For optimization, the state features $\hat{x}$ include 2 continuous features and 3 discrete features. The continuous features are taken from the belief state and are (1) the probability that the top callee is correct, and (2) the probability that the top callee's type of phone (office or cell) is correct. The discrete features are taken from the hand-designed dialog controller and are (1) the number of phone types the top callee has {none, one, two}, (2) whether the top callee is ambiguous {yes, no}, and (3) whether confirmation has yet been requested for the top callee {yes, no}.

Finally, a simple reward function was created which assigns -1 per system action plus +/-20 for correctly/incorrectly transferring the caller at the end of the call.

Optimization was performed on "POMDP" and "HC+POMDP" following the summary method (Williams and Young 2007b). An appendix provides complete details of the optimization; in sketch, K synthetic dialogs were generated by randomly choosing allowed actions. The space of state features was quantized into small regions, and a transition and reward function over these regions

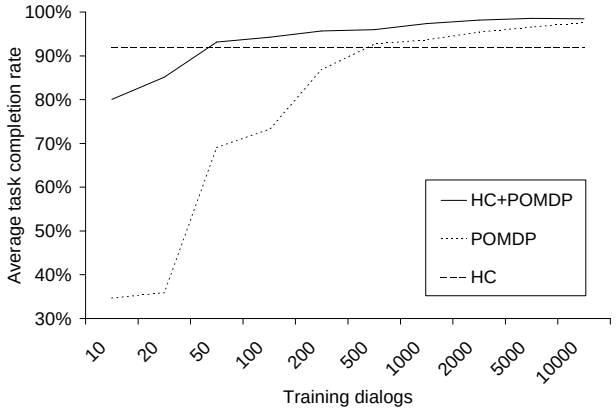

Figure 1: *Number of training dialogs K vs. task completion rate for the HC+POMDP, POMDP, and HC dialog managers.*

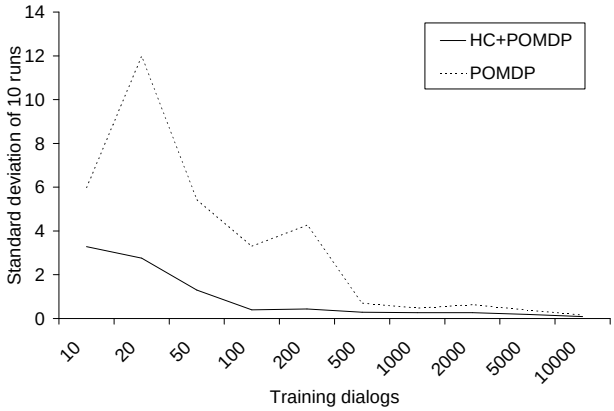

Figure 2: *Number of training dialogs vs. standard deviation in reward measured over 10 independent runs for the HC+POMDP and POMDP dialog managers.*

were estimated by frequency counting, applying some smoothing to mitigate data sparsity. Straightforward value iteration was then applied to the estimated transition and reward functions to produce a value function $\hat{Q}(\hat{x}, \hat{a})$. The optimization procedure, simulation environment, state features, and action set were identical for "POMDP" and "HC+POMDP": the only difference was whether the set of allowed actions was constrained or not.

Results

Using the system described above, optimization was conducted for various numbers of K dialogs for "POMDP" and "HC+POMDP", ranging from $K = 10$ to $K = 10,000$. After optimization, each policy was evaluated in simulation for 1000 dialogs to find the average return, average task completion rate, and average dialog length. The simulation environments for optimization and evaluation were identical. For each value of K, this whole process (optimization and evaluation) was run 10 times, and the results of the 10 runs were averaged. 1000 simulated dialogs were also run with the baseline "HC", using the same simulation environment.

Results for task completion rate are shown in Figure 1. As the number of training dialogs increases, performance of both POMDP and HC+POMDP increase to roughly the same asymptote.[1] With sufficient training dialogs, both POMDP and HC+POMDP are able to out-perform the baseline. However, HC+POMDP reaches this asymptote with many fewer dialogs. Moreover, inspection of the 10 runs at each value of K showed that the HC+POMDP policies were significantly more consistent: Figure 2 shows that the standard deviation of the average total reward per dialog over the 10 runs is lower for HC+POMDP than for POMDP.

[1]Dialog length for all systems was also measured: it hovered very close to 4 system turns and did not show any clear trend.

These results verify that, in dialog simulation at least, incorporating a POMDP into a conventional dialog system increases performance. Moreover, when compared to a pure POMDP, this method reduces training time and yields more consistent results vs. a pure POMDP. In other words, not only does this method combine the strengths of the two methods, it also reduces optimization time and less often produces spurious policies.

One of the policies created using with this method trained on 10,000 simulated dialogs was installed in our internal phone system, and it is now available alongside the baseline system. Its response time is essentially identical to the baseline system (2-3s). A webpage is available which shows the belief state and action selection running in real-time, and we have started to collect usage data. Figure 3 shows an example conversation illustrating operation of the method in detail and showing screenshots of the webpage.

Conclusions

This paper has presented a simple method to integrate expert knowledge into POMDP optimization in the context of spoken dialog systems. A POMDP belief state and a conventional controller run in parallel, and the conventional controller is augmented so that it nominates a set of acceptable actions. The POMDP chooses an action from this limited set. The method naturally accommodates compression (demonstrated here using the "summary" technique), which enables the method to scale to non-trivial domains – here a voice dialer application covering 50,000 listings. Simulation experiments demonstrate that the method outperformed our existing baseline dialer, while simultaneously requiring less training data than a classical POMDP.

We hope this method moves POMDPs an important step closer to commercial readiness for dialog systems. We are

also interested to explore whether this technique is useful in other domains.

References

Doshi, F., and Roy, N. 2007. Efficient model learning for dialog management. In *Proc of Human-Robot Interaction (HRI), Washington, DC, USA*.

Doshi, F.; Pineau, J.; and Roy, N. 2008. Reinforcement learning with limited reinforcement: Using Bayes risk for active learning in POMDPs. In *Proc 10th International Symposium on AI and Mathematics, Fort Lauderdale, FL, USA*.

Roy, N.; Pineau, J.; and Thrun, S. 2000. Spoken dialog management for robots. In *Proc Association for Computational Linguistics (ACL), Hong Kong*, 93–100.

Williams, J., and Young, S. 2007a. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech and Language* 21(2):393–422.

Williams, J., and Young, S. 2007b. Scaling POMDPs for spoken dialog management. *IEEE Trans. on Audio, Speech, and Language Processing* 15(7):2116–2129.

Williams, J. 2007. Using particle filters to track dialogue state. In *Proc IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), Kyoto, Japan*.

Young, S.; Williams, J.; Schatzmann, J.; Stuttle, M.; and Weilhammer, K. 2006. The hidden information state approach to dialogue management. Technical Report CUED/F-INFENG/TR.544, Cambridge University Engineering Department.

Young, S.; Schatzmann, J.; Thomson, B. R. M.; KWeilhammer; and Ye, H. 2007. The hidden information state dialogue manager: A real-world POMDP-based system. In *Proc NAACL-HLT, Rochester, New York, USA*.

Zhang, B.; Cai, Q.; Mao, J.; and Guo, B. 2001. Planning and acting under uncertainty: A new model for spoken dialogue system. In *Proc Conf on Uncertainty in Artificial Intelligence (UAI), Seattle, Washington*, 572–579.

Appendix: Policy Estimation

To estimate the policy, first a set of K synthetic dialogs are sampled by running the simulation environment in a generative mode and choosing an action from the set of allowed actions uniformly at random. The sequence of feature vectors, action mnemonic, and rewards is logged so each simulated dialog is a sequence $(\hat{x}_0, \hat{a}_0, r_0, \hat{x}_1, \hat{a}_1, r_1, \ldots, \hat{x}_n)$.

Next we define a distance metric over feature vectors $D(\hat{x}_i, \hat{x}_j)$ as the maximum distance of an element l in the feature vectors:

$$D(\hat{x}_i, \hat{x}_j) = \max_l (d(\hat{x}_i(l), \hat{x}_j(l)))$$

where

$$d(\hat{x}_i(l), \hat{x}_j(l)) = \begin{cases} |\hat{x}_i(l) - \hat{x}_j(l)|, & \text{if } x(l) \text{ continuous;} \\ 1, & \text{if } x(l) \text{ discrete, } \hat{x}_i(l) \neq \hat{x}_j(l); \\ 0, & \text{if } x(l) \text{ discrete, } \hat{x}_i(l) = \hat{x}_j(l). \end{cases}$$

It is assumed that the continuous elements are probabilities, so $\max_{\hat{x}_i, \hat{x}_j} D(\hat{x}_i, \hat{x}_j) = 1$.

Next we build a set of template points $\mathcal{Y}$ using a simple nearest-neighbor procedure. We consider each $\hat{x}$ of each sampled dialog in order; if the $\min_{\hat{y} \in \mathcal{Y}} D(\hat{x}, \hat{y}) > \epsilon$, we add $\hat{x}$ to $\mathcal{Y}$, where ϵ is a parameter we define.

Our goal is to estimate a transition probability over these template points $P(\hat{y}'|\hat{y}, \hat{a})$. With sufficient data, each template point/action pair $(\hat{y}, \hat{a})$ would be supported by many samples $(\hat{x}, \hat{a})$ in the synthetic dialog data. However, in practice often many $(\hat{y}, \hat{a})$ have few supporting samples, and early experiments showed that this caused substantial misestimation of $P(\hat{y}'|\hat{y}, \hat{a})$ which led to spurious policies.

As a result, we developed a smoothing technique. The intuition is to grow the set of samples for each template point by adding samples which are further away, but to *discount* the contribution of those samples which are further away from the template point. In detail, for each $\hat{y} \in \mathcal{Y}$ and for each $\hat{a} \in \mathcal{A}$, we find all $(\hat{x}, \hat{a}, \hat{x}')$ tuples in the synthetic data such that $D(\hat{x}, \hat{y}) \leq \epsilon$. We then find $\hat{y}' = \arg\min_{\hat{y}' \in \mathcal{Y}} D(\hat{x}', \hat{y}')$, and compute a weight $w(\hat{x}, \hat{y})$

$$w(\hat{x}, \hat{y}) = 1 - D(\hat{x}, \hat{y})$$

and add $w(\hat{x}, \hat{y})$ to a set $\mathcal{W}_{\hat{y}, \hat{a}, \hat{y}'}$. We define $\mathcal{W}_{\hat{y}, \hat{a}} = \bigcup_{\hat{y}'} \mathcal{W}_{\hat{y}, \hat{a}, \hat{y}'}$

If $\mathcal{W}_{\hat{y}, \hat{a}}$ contains fewer than γ entries, we consider it underestimated, where γ is a parameter we define. In this case, we increase ϵ and repeat the process above until either $|\mathcal{W}_{\hat{y}, \hat{a}}| \geq \gamma$, or ϵ reaches ϵ_{max}, whichever comes first. ϵ_{max} is another parameter we define.

Finally we estimate $P(\hat{y}'|\hat{y}, \hat{a})$ as

$$P(\hat{y}'|\hat{y}, \hat{a}) = \frac{\sum_{w \in \mathcal{W}_{\hat{y}, \hat{a}, \hat{y}'}} w}{\sum_{\hat{y}'} \sum_{w \in \mathcal{W}_{\hat{y}, \hat{a}, \hat{y}'}} w}$$

The reward function over template points $R(\hat{y}, \hat{a})$ is estimated in an analogous way, by weighting the observed rewards.

The transition and reward functions $P(\hat{y}'|\hat{y}, \hat{a})$ and $R(\hat{y}, \hat{a})$ now form a standard MDP model and value iteration can be applied to compute $\hat{Q}(\hat{y}, \hat{a})$. At run-time, for a given feature vector $\hat{x}$, we find the closest template point $\hat{y} = \arg\min_{\hat{y} \in \mathcal{Y}} D(\hat{x}, \hat{y})$ and use its $\hat{Q}(\hat{y}, \hat{a})$. After experimentation we ultimately used parameters of $\epsilon = 0.01$, $\epsilon_{max} = 0.2$, and $\gamma = 100$.

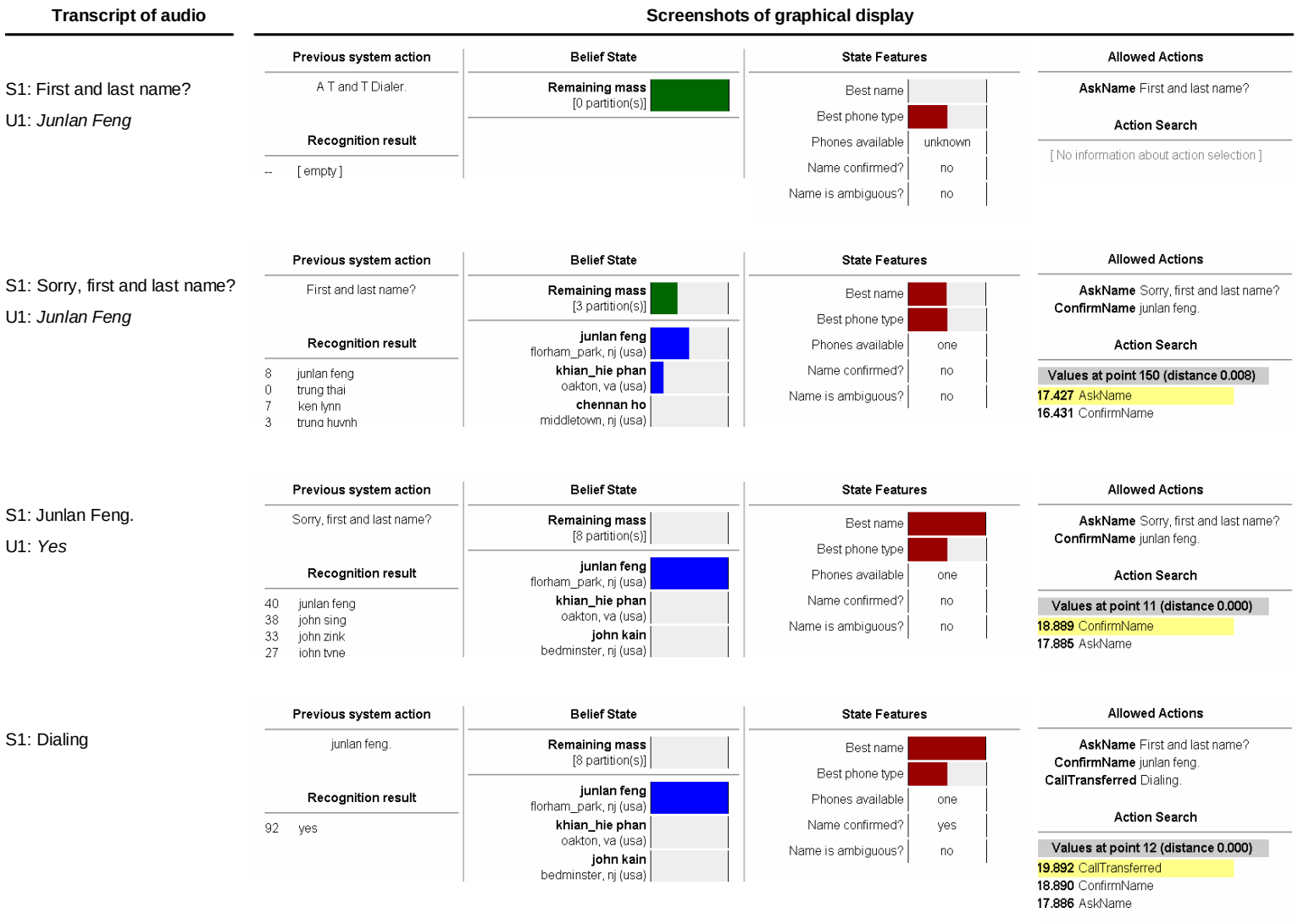

Figure 3: *Illustration of operation of the HC+POMDP dialog manager. The first column shows a transcript of the audio. The second column shows the previous system action and the first few speech recognition hypotheses from the N-Best list, with their confidence scores. The third column shows a portion of the full POMDP belief state $b(s)$, which is a distribution over all possible 50,000 callees. The fourth column shows the state features $\hat{x}$ used for action selection. The top of the final column shows the list of allowed actions, with the full action a in normal typeface and the action mnemonic $\hat{a}$ in bold. Finally the bottom of the final column shows the value estimated by the optimization for each summary action given the current state features $\hat{Q}(\hat{x}, \hat{a})$. The highlighted row indicates the maximum $\hat{Q}$ value, whose action mnemonic is selected for output to the user.*