User Model User-Adap Inter (2007) 17:71–91
DOI 10.1007/s11257-006-9020-7

ORIGINAL PAPER

Personalizing influence diagrams: applying online learning strategies to dialogue management

David Maxwell Chickering · Tim Paek

Received: 1 November 2005 / Accepted in revised form: 29 June 2006 /
Published online: 13 December 2006

Abstract We consider the problem of adapting the parameters of an influence diagram in an online fashion for real-time personalization. This problem is important when we use the influence diagram repeatedly to make decisions and we are uncertain about its parameters. We describe learning algorithms to solve this problem. In particular, we show how to modify various explore-versus-exploit strategies that are known to work well for Markov decision processes to the more general influence-diagram model. As an illustration, we describe how our techniques for online personalization allow a voice-enabled browser to adapt to a particular speaker for spoken dialogue management. We evaluate all the explore-versus-exploit strategies in this domain.

Keywords Personalization · Influence diagrams · User-model adaptation · Planning · Dialogue management · Speech recognition

1 Introduction

Statistical and probabilistic methods for user modeling have become increasingly popular due in part to two advantages they afford over the traditional knowledge-base approach: first, they provide a natural quantitative framework for representing uncertainty about user modeling, and second, they can be trained quickly on user data (Zukerman and Albrecht 2001). Probabilistic graphical models, such as Bayesian networks, have been successfully applied to infer user goals in several domains (e.g., Albrecht et al. 1998; Horvitz et al. 1998). In this paper, we describe learning and

D. M. Chickering (✉) · T. Paek
Microsoft Research,
One Microsoft Way, Redmond, WA 98052, USA
e-mail: dmax@microsoft.com

T. Paek
e-mail: timpaek@microsoft.com

inference algorithms for a particular graphical model, the *influence diagram*, which is a generalization of a Bayesian network. The model represents both the environment and preferences of a user to help make decisions. Influence diagrams can either be constructed by an expert (typically with the aid of a decision analyst), or learned from data. As described by Heckerman (1995), supervised learning techniques for "ordinary" Bayesian networks apply easily (i.e., with little modification) to learn the structure and parameters of an influence diagram.

As an influence diagram is used over time, we would like to use the resulting observed data to improve the model. This is particularly important when starting out with a "baseline" influence diagram that is constructed (either from data or from an expert) to reflect the behaviors and preferences of a general population of users; when that model is used repeatedly by a *particular* user, the model should adapt to better reflect the specific behaviors and preferences of that user. Online adaptation is also important when we are generally uncertain about the parameter values and/or when the domain can be changing over time (i.e., there is concept drift).

In this paper, we consider scenarios where we use an influence diagram to make repeated decisions that maximize the long-term expected utility of a particular user over time. Our problem is similar to the classic n-armed bandit problem in the sense that if we always "exploit" our current knowledge to maximize immediate reward, we may never learn about the specific behaviors of the user that can lead to alternative strategies that may be superior in the long term. Each "pull" of a bandit, however, corresponds in our problem to deriving optimal actions within an influence diagram. For this reason, we call our problem the *Bandit-ID* (Influence Diagram) problem. In most cases, identifying the optimal set of actions to take in a Bandit-ID problem is intractable due to the uncertainty in the model parameters.

Markov decision processes (MDPs) have traditionally been used to model domains in which we need to deal with an explore-versus-exploit tradeoff. An MDP is a special case of an influence diagram in which there is repeated structure used to make a sequence of identical (state-dependent) decisions. In this paper, we will apply some of the same *strategies* used to learn MDPs to come up with new algorithms for online personalization of influence diagrams. Online algorithms for learning MDPs exist (see Boutilier et al. 1999), but these algorithms rely on the repeated structure of state-transition probabilities that does not generally occur in an influence diagram. Although there may be influence-diagram extensions to some of these MDP learning algorithms, these extensions are not obvious, and we believe this is the first treatment of online personalization in the more general setting.

There are three main contributions of this paper. First, in Sect. 3, we present the details of the Bandit-ID problem. Second, in Sect. 4, we describe a number of standard explore-versus-exploit strategies found in the MDP literature, and we describe how they can be implemented and used to solve Bandit-ID problems. Finally, we describe results from a real-world user-modeling problem that is modeled as a Bandit-ID problem. In particular, in Sect. 5 we describe an influence diagram that controls a speech-enabled web browser that adapts to the user, and in Sect. 6 we compare the various explore-versus-exploit strategies in this domain.

2 Background

In this section, we provide background material relevant to the rest of the paper. We denote a variable by an upper case token (e.g., A, Θ) and a state or value of that variable by the same token in lower case (e.g., a, θ). We denote sets of variables with bold-face capitalized tokens (e.g., $\mathbf{A}, \mathbf{\Theta}$) and corresponding sets of values by bold-face lower case tokens (e.g., $\mathbf{a}, \boldsymbol{\theta}$).

An *influence diagram* is a graphical model used to make a sequence of one or more decisions. The model is defined over a domain consisting of three types of variables: decision variables $\mathbf{D}$, chance variables $\mathbf{U}$, and value variables $\mathbf{V}$. The influence diagram also contains a single utility function $f_{\text{util}}(\mathbf{V})$ that is a deterministic function of all of the value variables. An influence diagram contains a set of parameters $\mathbf{\Theta}$ that characterize the conditional distributions of the non-decision variables. In particular, the diagram defines the probability distribution $p(\mathbf{U}, \mathbf{V}|\mathbf{D}, \mathbf{\Theta})$ via the local distributions stored within the non-decision nodes:

$$p(\mathbf{U}, \mathbf{V}|\mathbf{D}, \mathbf{\Theta}) = \prod_{X \in \mathbf{U} \cup \mathbf{V}} p(X|\mathbf{Pa}(X), \mathbf{\Theta}_X)$$

where $\mathbf{Pa}(X)$ denotes the set of parents for node X, and where $\mathbf{\Theta}_X$ denotes the subset of parameters in $\mathbf{\Theta}$ that define the local distribution of X. Note that the parent set $\mathbf{Pa}(X)$ can contain decision nodes from $\mathbf{D}$. Parents of a decision node D are the nodes that will be observed at the time decision D is made.

Our definition deviates from the traditional definition of an influence diagram (see, e.g., Howard and Matheson 1981) because we allow there to be multiple value nodes that are not necessarily deterministic. Our use of multiple stochastic value nodes is simply an optimization to allow efficient representation of a factored utility function (e.g., the utility is the sum of all the values nodes), and has been used by other researchers such as Tatman and Shachter (1990) and Lauritzen and Nilsson (2001). In Fig. 1, we show an example of such an influence diagram. The circular nodes represent the chance variables, the square nodes represent the decision variables, and the diamond nodes represent the value variables.

If the parameters $\mathbf{\Theta}$ of an influence diagram are known with certainty, we can apply well-studied inference techniques to "solve" for the optimal sequence of decisions represented in that diagram (e.g., Howard and Matheson 1981; Tatman and Shachter 1990; Shachter and Peot 1992; Cooper 1993).[1] In particular, corresponding to each setting $\boldsymbol{\theta}$ of the parameters is an optimal *policy* $\mathcal{P}(\boldsymbol{\theta})$ that prescribes, for each decision node in the influence diagram, what the best choice is as a function of the values of the observed variables. In practice, the policy $\mathcal{P}(\boldsymbol{\theta})$ is not constructed explicitly, but rather as each decision needs to be made, an inference algorithm is run to determine the best action to take. For example, given any set of observations $\mathbf{O} = \mathbf{o}$, where $\mathbf{O} \subset \{\mathbf{U} \cup \mathbf{V} \cup \mathbf{D}\}$, we run the inference algorithm to determine the value d for decision node $D \in \mathbf{D}$ that maximizes $E(f_{\text{util}}(\mathbf{V})|\mathbf{O} = \mathbf{o}, D = d)$.

[1] We note that many algorithms that "solve" an MDP (such as value iteration) are particular instances of inference algorithms that take advantage of the repeated (and sometimes infinite) structure of an MDP.

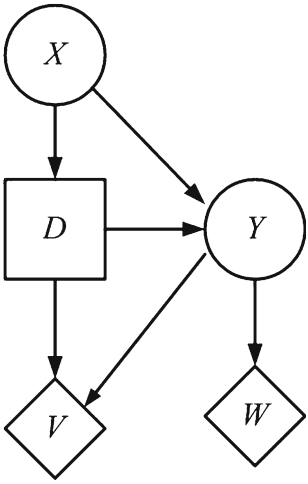

Fig. 1 An example of an influence diagram. The parameters of the model are not shown

3 The Bandit-ID problem

For the Bandit-ID problem, we assume that the parameters of the influence diagram are not known with certainty, but rather that we have a probability distribution $p(\mathbf{\Theta}|\cdot)$. The goal is to maximize the *long-term* utility of using the model in an online fashion. As described in Sect. 1, the Bandit-ID problem is relevant when we have an influence diagram, perhaps with known parameters, that reflects a population of users, and we would like to adapt that model to a *particular* user. In this case, we express the uncertainty about how the particular user deviates from average using the probability distribution $p(\mathbf{\Theta}|\cdot)$; if the parameters of the population model are themselves uncertain, we can introduce further uncertainty by increasing the variance of this distribution.

We assume that the influence diagram is going to be used repeatedly, and we call each use of the influence diagram a *trial*. For example, consider an influence diagram used to help a person make a sequence of decisions about how to get home from work. Using the model we might decide to take a detour instead of the highway, and after making this decision, we need to decide which detour to follow. In this example, the trial corresponds to a single trip home. Note that a trial in a Bandit-ID problem, which can involve multiple decisions, corresponds to a *single* pull in the n-armed bandit problem.

After each trial, we update the parameter distribution $p(\mathbf{\Theta}|\cdot)$ given the resulting observed values (e.g., it took an hour to get home and there was above-average traffic on the particular detour). We use $\mathbf{D}_i$, $\mathbf{U}_i$, and $\mathbf{V}_i$ to denote the decision, chance, and value variables, respectively, in the *ith* trial. We use $\mathbf{D}^n = \cup_{i=1}^n \mathbf{D}_i$ to denote the union of all decision variables up through the *nth* trial; we use $\mathbf{U}^n$ and $\mathbf{V}^n$ to denote the corresponding unions for the chance and value variables, respectively.

As shown by Heckerman (1995), learning the parameters of an influence diagram from observed trial data is a straightforward extension to the problem of learning the parameters of a Bayesian network; the only significant difference is that the joint distribution of interest in an influence diagram is one that is conditioned on a set of decisions. By making the standard assumptions for learning Bayesian networks (see Heckerman 1996) we can compute the posterior distribution

$$p(\boldsymbol{\Theta}|\mathbf{U}^n, \mathbf{D}^n, \mathbf{V}^n)$$

in closed form, and use this distribution to choose the policy for the next $(n+1)$ trial. In particular, we can extract the MAP parameter values $\hat{\boldsymbol{\theta}}$ from this distribution, and as shown by Heckerman (1996) for Bayesian networks, these values render the next trial independent of the past data:

$$p(\mathbf{U}_{n+1}, \mathbf{V}_{n+1}|\mathbf{D}_{n+1}, \mathbf{U}^n, \mathbf{D}^n, \mathbf{V}^n) = p(\mathbf{U}_{n+1}, \mathbf{V}_{n+1}|\mathbf{D}_{n+1}, \hat{\boldsymbol{\theta}})$$

What this means is that after n trials, we can identify the (locally) optimal policy for trial $n+1$ by performing the well-known inference algorithms under the assumption that the parameters are known to be equal to their MAP values.

What makes the Bandit-ID problem challenging is that, as with the well-studied n-armed bandit problem (see, e.g., Berry and Fristedt 1985), we are faced with the classic "explore versus exploit" problem from planning algorithms: if we are only interested in maximizing our expected return on the *next* trial, it is easy to see that the optimal policy is to "exploit" our knowledge and take actions that result in the highest expected immediate reward; that is, simply set the parameters to their MAP values and solve the resulting (parameterized) influence diagram. Given $k > 1$ remaining trials, however, it may be better to "explore" by experimenting with sub-optimal actions in order to gain more information about uncertain parameters.

4 Explore-versus-exploit strategies for selecting actions

In this section, we describe a number of heuristic strategies that can be applied to the Bandit-ID problem to hopefully yield high long-term utility. In Sect. 6, we compare all of these strategies in our real-world application.

4.1 Thompson

The first strategy, originally due to Thompson (1933), is the *Thompson* strategy which prescribes that in each trial, policies are chosen stochastically according to the probability of being optimal. Wyatt (1997) applied the Thompson strategy to a variety of bandit problems and showed that it is superior to many other simple explore-versus-exploit strategies found in the reinforcement-learning literature. Dearden et al. (1998) applied the Thompson strategy for action selection in a model-free reinforcement-learning problem, using a Bayesian approach to Q-learning. Dearden et al. (1998) recognized that although deriving the probability that each decision is optimal may be difficult, sampling decisions according to this distribution is simple.

We now describe the insight of Dearden et al. (1998), and show how the Thompson strategy is particularly easy to implement within the Bandit-ID setting. Recall from Sect. 2 that if the parameter values $\boldsymbol{\theta}$ of an influence diagram are known, then we perform optimally at each step by simply performing the unique optimal policy $\mathcal{P}(\boldsymbol{\theta})$ in each trial. It follows that the probability any particular policy $\mathcal{P}$ is optimal is equal to the total probability mass over the set of parameters that result in $\mathcal{P}$ being optimal. This means that by simply sampling the parameters from the posterior distribution and solving the resulting influence diagram, we are selecting policies based on the probability that they are optimal. Furthermore, it is particularly easy to sample from $p(\boldsymbol{\Theta}|\cdot)$ when the variable-specific parameter distributions are modeled as Dirichlet or

normal-Wishart distributions, which are the standard choices for discrete and continuous variables, respectively, when learning Bayesian networks. In short, the Thompson strategy repeats the following steps:

1. Sample an instance of the model parameters $\boldsymbol{\theta}$ from their posterior distribution.
2. Solve the resulting influence diagram to determine the best policy.
3. Update the parameter distribution given the observed data.

Note that although the Thompson strategy implicitly imposes a distribution over the actions we take, we never need to construct this distribution explicitly. The randomness of the algorithm comes entirely from sampling the model parameters because once these parameters are known, the optimal actions are well defined through inference in the model.

4.2 Interval

The second strategy is the *interval-estimation* strategy—or *Interval* for short—of Kaelbling (1993), which Wyatt (1997) found to consistently outperform all other strategies on bandit problems. The strategy calculates a 95% confidence interval around the expected utility of each action, and then chooses the action with the highest upper bound. We can implement this strategy in a Bandit ID in a similar fashion as the Thompson strategy, except that instead of sampling the parameters once, we sample them many times and collect the resulting expected utilities for each action. With these samples, we construct an empirical distribution over the expected utilities, and choose the action with the highest bound on the 95% confidence interval. In our experiments, we used only 20 samples because we wanted the strategy to be able to run in real time and because 20 samples worked just as well as 100 samples.[2]

4.3 Exploit

The *Exploit* strategy uses the MAP values of the posterior distribution to parameterize the model. After solving the influence diagram at each step, we simply perform the action that maximizes the immediate reward. With this strategy we adapt the parameter distributions as we observe data, but we make decisions based on the MAP values only.

4.4 Epsilon

The *Epsilon* strategy is the same as Exploit, except that with some probability ϵ, we choose a non-optimal action uniformly at random. We used $\epsilon = 0.1$ in our experiments; this value is known to perform well on other explore-versus-exploit tasks (see, e.g., Sutton and Barto 1998).

4.5 Boltzmann

In the *Boltzmann* strategy, we again use the MAP values of the posterior distribution to parameterize the model. The Boltzmann strategy is to sample actions according to the following Gibbs distribution:

[2] To get the upper bound, we used the second-highest utility value in the sample. Technically, this is the upper bound on the 90% confidence interval, but the strategy was indistinguishable from using the 95% confidence interval when we used 100 samples.

$$p(\text{choose } a) \propto \mathrm{e}^{U(a)/T}$$

where $U(a)$ is the expected utility of action a, and where T is a temperature parameter that is decreased over time. For high values of T, the strategy chooses actions almost uniformly; as T decreases, the strategy converges to the Exploit strategy. For our experiments, we started $T = 1600$ and then decreased to 1 by dividing in half on each iteration. For more discussion on this strategy, see Kaelbling et al. (1996).

5 Browser application

In this section, we describe how we used a Bandit ID to facilitate user-model personalization for a voice-enabled browser called *Accessibility Browser*. The browser functions just like any other web browser, except that it responds to speech commands and can engage in a repair dialogue if it does not understand an utterance. We used the Microsoft Speech API (SAPI) as the speech-recognition engine.

We instrumented Accessibility Browser to respond to 20 commands. 17 of these commands had no arguments:

1. ADD-TO-FAVORITES: Adds the current page to the favorites list.
2. FORWARD: Moves forward in the history list.
3. BACK: Moves back in the history list.
4. FONT-DOWN: Decreases the font size.
5. FONT-UP: Increases the font size.
6. HOME: Moves to the home page.
7. PAGE-DOWN: Scrolls down one screen.
8. PAGE-UP: Scrolls up one screen.
9. READ: A text-to-speech engine reads the page out loud.
10. REFRESH: Reloads the current page.
11. SCROLL-DOWN: Continuously and slowly scrolls down.
12. SCROLL-UP: Continuously and slowly scrolls up.
13. STOP: Stops scrolling.
14. SHOW-FAVORITES: Brings up a window with the favorites list. Each link in the list is labeled with an integer.
15. HIDE-FAVORITES: Removes the window with the favorites list.
16. SHOW-NUMBERS: Adds a number next to every link, button, and edit box on the page.
17. HIDE-NUMBERS: Removes the numbers that were added from SHOW-NUMBERS.

The remaining three commands are active when either the numbers are showing after a SHOW-NUMBERS command, or when the the window with the favorites list is being shown. These commands take an integer argument, and specify an action to be taken on a numbered item:

1. CLICK-N: Performs a click on the named button.
2. FILL-N: Move focus to the named edit box.
3. GOTO-N: Navigate to the named link.

Each command has one or more English phrases that we specify in a grammar file that is used by SAPI. For example, we can activate the BACK command by saying either "back" or "go back". The grammar file also allows us to specify that the commands with arguments have integer arguments, and thus SAPI can recognize "go to fourteen" as mapping to the GOTO-N command with argument value 14.

When the user speaks into a microphone, the speech signal is sent to SAPI, and SAPI returns a number of speech features. Of significant importance is the *top-n* list returned by SAPI: the API returns a sorted list of commands with corresponding confidence scores. For example, suppose the user says "font up" in a noisy environment. SAPI might return the list FONT-UP, FONT-DOWN, PAGE-UP, and BACK, with corresponding confidence scores of 0.8, 0.4, 0.3, and 0.1 (confidence scores are not probabilities). Due to the fact that there are multiple ways of issuing the same command, the top-n list can contain duplicate commands. For example, BACK might be both the first and second command in the top-n list due to "back" and "go back" being the two most likely recognitions according to SAPI.

When the user first issues a spoken command to the browser, the system analyzes the speech features returned by SAPI and makes a decision between three different actions: it can perform the *Ignore* action (which is appropriate when background noise is sent to SAPI), it can perform the *DoTop* action, which means that it proceeds with the top command in the top-n list, or it can engage in a repair dialogue. There are two choices for the first-round repair: the browser can perform a *Repeat* repair, where it asks the user to repeat the command, or the browser can perform a *Confirm* repair, where it presents the user with a list of the likely commands with corresponding integers. The confirmation list always contains the elements "none of the above" and "I didn't say anything" as the last two choices.

If a repair dialogue is initiated by the system, the user must issue another command. In the case of a *Repeat* he repeats the same command again, and in the case of a *Confirm* he names the appropriate integer. After processing the second command, the browser once again analyzes the resulting speech features and decides between taking the *DoTop* action or continuing the repair dialogue. If the original repair was *Confirm*, and if the best SAPI recognition is the integer corresponding to "none of the above" or "I didn't say anything", then taking the top action in the second step results in an apology to the user. If the browser decides to engage in a second repair, it must perform a dialogue-terminating action after processing the next command. After analyzing the speech features it can either perform a *DoTop* action, or it can perform a *Bail* action in which the dialogue ends with an apology.

We implemented Accessibility Browser using a Bandit ID to decide what actions to take in each step of the dialogue process described above. In related research, spoken-dialogue researchers have noted that speech interaction entails sequential decision making that can be modeled as an MDP (Young 2000; Singh et al. 2002). If user intention is included as an unobservable variable, then that decision making is modeled as a partially observable MDP (Roy et al. 2000). We found an influence diagram to be a more natural model choice for a number of reasons. First, because users are not likely to put up with long dialogues, the interaction was naturally finite and very short in length (at most three steps in our implementation). For example, asking a user to repeat a command 10 times before acting is not going to be acceptable. Second, the nature of the interaction at each time step varied significantly. As we show below, both the variables for predicting success and the overall behavior of the reward function were different in each step. Third, we found important relationships among variables

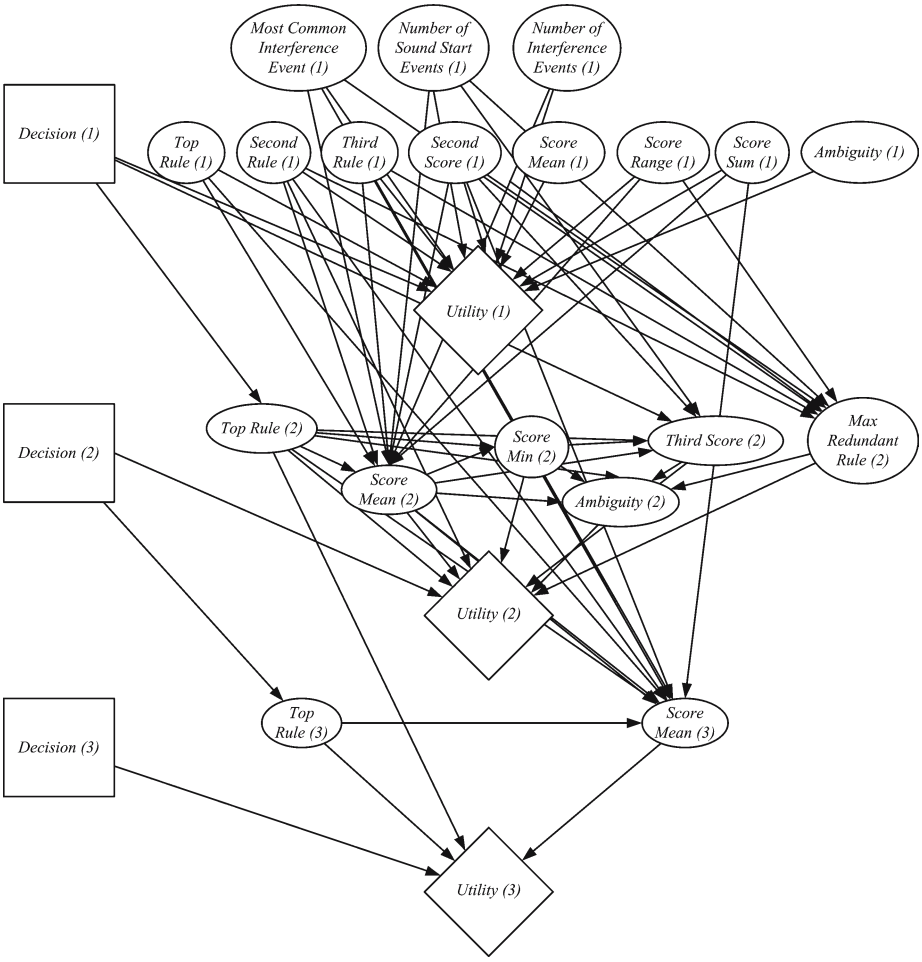

Fig. 2 The influence diagram used for Accessibility Browser. We omit edges directed into the decision nodes

that were not Markovian; in particular, we found that the second-stage variables did not render the third-stage variables independent of the first-stage variables (see Fig. 2). Finally, for repair dialogues in general, the set of actions that are appropriate at each time step may be varied. For example, we did not deem it appropriate for the browser to ignore the user after engaging in a repair. Our approach also differs from previous work in that we adapt the policy in an online fashion for personalization, as described in the previous section, so that the system can be customized to individual users.

5.1 Learning the baseline model

In this section, we describe a novel supervised-learning technique for automatically constructing a baseline influence diagram for making decisions about what action to take in each step of the dialogue system described above. Our goal was to create a general model that includes speech features that are useful for working with a wide

variety of voices. In Sect. 6, we then show how the reinforcement learning techniques from Sect. 4 can be used to adapt the parameters of this model to a particular user.

We generated synthetic training data for the supervised-learning task using several text-to-speech (TTS) voices to dictate commands. We varied all the possible parameters of these voices (such as pitch, rate, and volume) and added to each utterance various levels and types of noise (such as crowd chatter and computer fans). From the simulation we generated roughly 20,000 dialogue sessions, where each session proceeded as follows. First, we randomly selected one of the possible commands, where "no command" was included as a possibility, and then "rendered" that command[3] with a random TTS voice (and parameters) with noise added, whereupon the speech engine recognized the utterance and generated SAPI events. If "no command" was selected, only the noise was generated. Next, we chose the action to perform at random: either *Ignore*, *DoTop*, *Repeat* or *Confirm*. If the action was not a repair, we recorded whether or not the system got the right command and ended the dialogue session. Otherwise, we proceeded to the next time step and rendered the appropriate utterance (for a *Confirm*, the voice would dictate the appropriate integer). As prescribed by the dialogue definition in the previous section, this process continued until at most three total interactions were made.

For each dialogue session, we recorded on the order of 45 speech features per time step (the number of features recorded depends on the time step), for up to 134 speech features for those sessions that lasted three steps. In the appendix, we list the details of all the features we used.

After logging all of our synthetic data, we learned the structure and parameters of an influence diagram for this domain using a version of the WinMine Toolkit (Chickering 2002). In particular, we first learned a decision tree for predicting immediate success or failure, at each time step, as a result of performing a *DoTop* or—in the case of the first time step—*Ignore*. For the learning process, the leaves in each of these trees contained independent multinomial distributions with Dirichlet parameter priors. Each of the three success/failure trees defined the function of the value node at each time step: for each context (i.e., for each leaf), the expected value of a *DoTop* or *Ignore* (first step only) is the probability of success times $V(\textit{success}) = 100$ plus the probability of failure times $V(\textit{failure}) = -100$. These specific maximum and minimum values are important only for defining the range of possible values for a repair action, which was a user-configurable constant in our system; for repair actions, the corresponding value $V(\textit{repair})$ was equal to this constant, regardless of the values of the chance nodes. After identifying the variables that were predictive of success in each time step, we eliminated all other variables from consideration. Finally, we learned a Bayesian network among all the (predictive) chance variables, the only constraint being that variables in time-step i could not be predictors of previous time steps. Once again, we learned decision-tree distributions for the local distributions in the Bayesian network. For the discrete variables, the leaves contained multinomial distributions with Dirichlet parameter priors, and for continuous variables, the leaves contained Gaussian distributions with normal-Wishart parameter priors. We did not learn any structure among the chance variables in the first time step, as these variables are always observed.

In Fig. 2, we show the structure of the influence diagram that resulted from our learning procedure. For readability, we leave implicit the "information arcs" that are

[3] For those commands with multiple phrases, we chose the phrase for the selected command randomly.

directed into the decision nodes. As mentioned above, we provide the details for all of these variables in the appendix. Briefly, from the figure, the "Rule" variables are discrete with values corresponding to the possible commands; for example, *Top Rule (2)* is the command corresponding to the highest-ranked recognition from the top-n list in the second step. In the case of a confirmation, the highest-ranked recognition might be "two", in which case the value for *Top Rule (2)* is the command whose text was listed in the second position in the confirmation list. The "Score" variables are corresponding continuous variables whose values are the confidence scores from SAPI. For example, *Second Score (1)* is the confidence score of the second item in the top-n list returned by SAPI in the first dialogue step. The "Ambiguity" variables indicate the number of distinct rules in the top-n list. The top three "Event" variables in the figure, whose definitions are given in the appendix, are useful for determining whether a sound event was simply background noise. Finally, *Max Redundant Rule (2)* indicates the cardinality of the most frequently occurring command in the top-n list.

The decision nodes in the model are represented in Fig. 2 as the squares on the left. The values of these nodes are the actions allowed at each step of the dialogue as described above. The value nodes are represented as the middle diamonds in Fig. 2. Finally, the total utility of an entire dialogue session is defined as the sum of the value nodes in the dialogue, except that if a time step was not entered (due to performing a *DoTop* or *Ignore* action), the corresponding value node did not contribute any value.

In Fig. 3, we show the decision-tree distribution for the node *Utility*(1). The histogram in each leaf node denotes, from top to bottom, the values for $p(failure)$, $p(repair)$, and $p(success)$. Note that the probability of the "repair outcome" is either zero or one, depending on whether or not a *Repair* action was taken. Assuming $V(repair) = -75$ (the value we use in the next section), we get the expected utilities from this tree by summing the products of each probability and the corresponding utility of -100, -75, and 100, respectively. For example, the top-most leaf in the figure, whose probabilities are 0.83, 0, and 0.17, has a corresponding expected utility of $-66 = -100 \times 0.83 - 75 \times 0 + 100 \times 0.17$. We see that the most important predictor of success in the first step of the dialogue is the value of the decision. Interestingly, we see that when predicting the success or failure from performing an *Ignore* action, the most informative check to make is whether or not the second rule in the top-n list corresponds to HIDE-NUMBERS; it turns out that if this is true, an *Ignore* action is much more likely to succeed. Examining the tree following the *DoTop* action, we see that the same test is important for determining the success of a *DoTop*; if the test is true, the *DoTop* action is much *less* likely to succeed. From this we can conclude that HIDE-NUMBERS often occurs as the second command in the top-n list when there is either no command or a hard-to-recognize command.

In Fig. 4, we show the conditional probability distribution for chance node *Score Mean (3)*, which is the average confidence score for the commands in the top-n list. The leaf nodes in this tree contain Gaussian distributions over the score. In the figure, the text within each leaf node designates the range between (1) the mean plus one standard deviation and (2) the mean minus one standard deviation. For example, the top-most leaf node contains the text "0.766 to 0.974", which results from a Gaussian distribution with a mean of 0.87 and a standard deviation of 0.104 (that is, 0.766 = 0.87 −0.104 and 0.974 = 0.87 + 0.104).

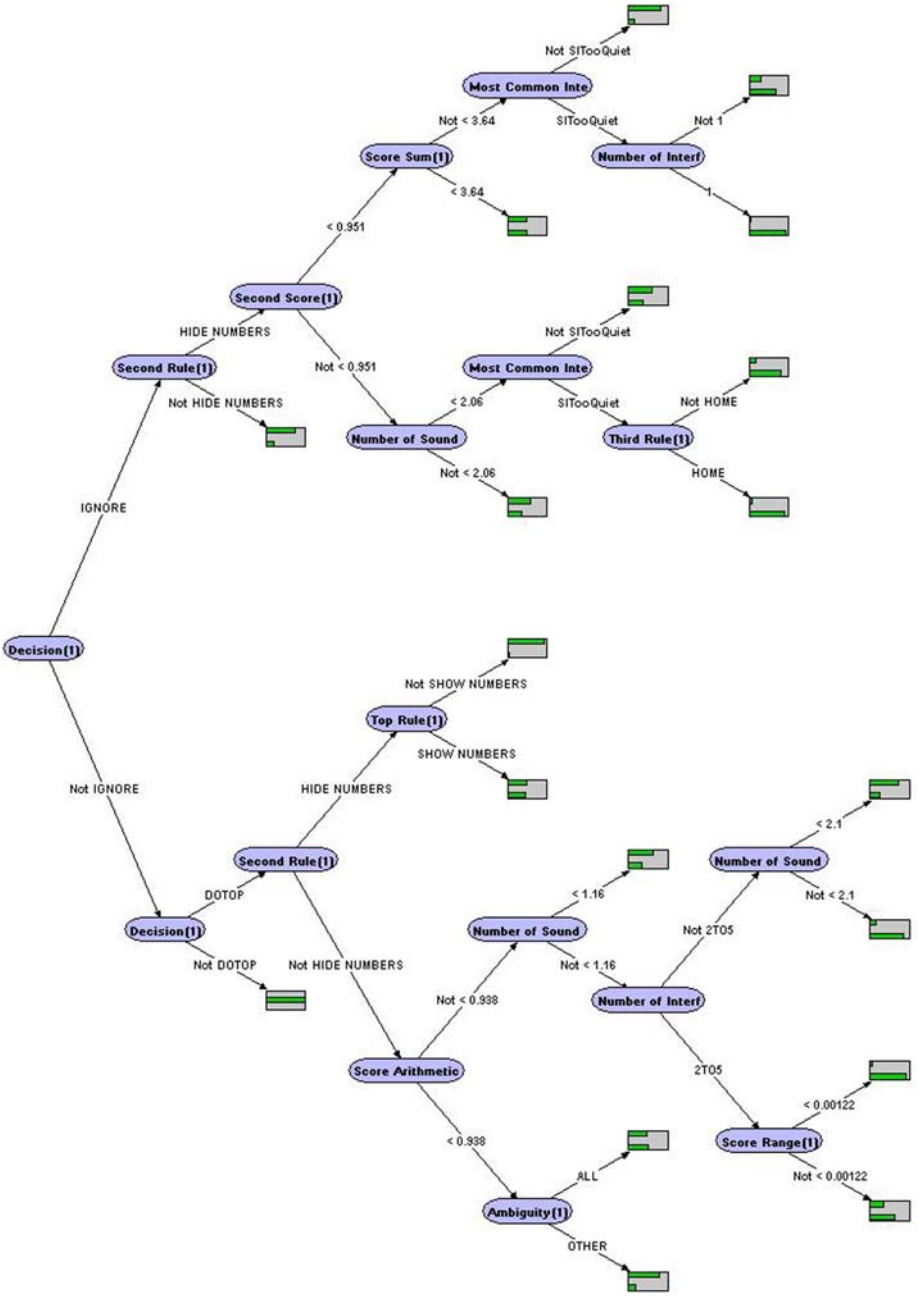

Fig. 3 The conditional probability distribution defining *Utility*(1). The histogram in each leaf node denotes from top to bottom: p(*failure*), p(*repair*), and p(*success*). To get the expected utility, we multiply these numbers by -100, -75, and 100, respectively, and then sum

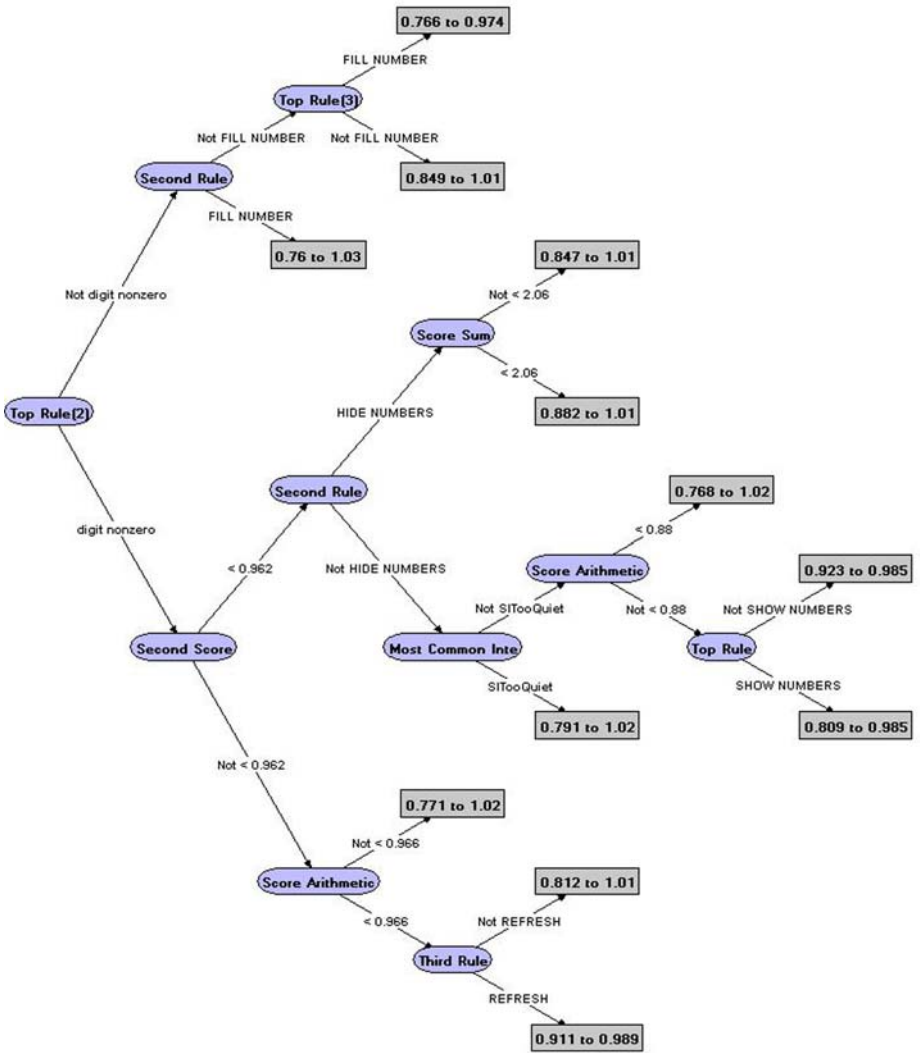

Fig. 4 Decision-tree distribution for the chance node *Second Score (2)*

5.2 Inference in the baseline model

As described in Sect. 2, we can run inference algorithms to solve for the optimal actions within a parameterized influence diagram. Intuitively, the process works as follows. When a user issues a command, all of the step-one variables are instantiated in the influence diagram. We can look up the total expected value for both the actions *DoTop* and *Ignore* directly from the decision tree for *Utility*(1) because, due to the fact that the dialogue terminates after these actions, the total value is equal to the step-one utility. To evaluate the expected value for each of the repair actions, however, we need to reason about what the future chance nodes might be; depending on the values of these chance nodes, we might make different decisions in the later steps.

Most of the standard inference algorithms described in the literature for solving influence diagrams require that the conditional distributions of the chance nodes are either all "complete tables" (i.e., all variables are discrete and there is a separate multinomial distribution for each possible combination of the parent values) or all conditional Gaussian distributions (i.e., all variables are continuous and are modeled as a linear functions of their parents). Because our influence diagram contains a combination of discrete and continuous variables, and because all conditional distributions are decision trees, we used a forward sampling approach instead.

For any dialogue step i, the inference algorithm works by, for each d_i, setting the decision node $D_i = d_i$ and then repeatedly sampling the chance nodes for dialogue step $i+1$, recursively solving for the best action in dialogue step $i+1$. For each sample, we collect the total utility, and we choose the action d_i that maximizes the average total utility among the samples. Note that whenever d_i is not a repair action, inference does not proceed to the next step. Because inference dominates the run time of the algorithm, and because we need the application to interact with a human in real time, we were careful to choose the minimum number of samples that were needed for the algorithm to converge. We found that by using 120 samples for dialogue step two and 60 samples for dialogue step three, the results of inference were the same as when many more samples were made.

Note that with our inference scheme, the time spent sampling at time step i is proportional to the number of nodes in that time step, and that the total number of samples needed grows exponentially with the number of utility nodes. For models with many more utility nodes, we might need to revise our inference strategy. For example, we could discretize all of the continuous variables, use table distributions, and then apply one of the standard inference algorithms as described in Sect. 2. Alternatively, we might use only one sample for dialogue steps larger than some fixed number of repair iterations into the future.

5.3 Personalizing the baseline model

The baseline model described above was learned using a diverse set of voices; the personalization problem is to adapt the model parameters to maximize utility for a *particular* user. We thus have precisely the Bandit ID problem described in Sect. 3, with the baseline influence diagram as our initial model. In the next section, we apply all of the reinforcement learning algorithms described in Sect. 4 to solve this problem.

6 Experiments

We compared the performance of the various explore-versus-exploit strategies from Sect. 4 within the Bandit-ID framework for the Accessibility Browser. Our goal was to determine how well each of the strategies would personalize the baseline model to a particular voice. Because the baseline model was learned using over 20,000 training sessions, the variances on the resulting parameter distributions were very small; to model our uncertainty about the baseline parameters for a *particular* voice, we reduced the equivalent-sample size in each distribution to five. This allowed the strategies to adapt the model parameters in a reasonable number of dialogue sessions.[4]

[4] We chose the value five because it worked well for adapting quickly to our own voices.

In addition to the strategies from Sect. 4, we included results from the following two default strategies:

- *DoTop* simply executes the top action recognized by SAPI at each step. Results from this strategy show how well the browser works without any repair dialogue.
- *Baseline* uses the MAP parameter values that result from the learning process described in Sect. 5.1, and does no adaptation of those parameters. Results from this strategy show how well the browser works without any personalization.

We evaluated how well the strategies adapted to individual voices as follows. We chose a set of 75 voices on which to run our experiments. In particular, we took the 5 TTS voices, then we chose 5 pitch settings and 3 rate settings to produce 75 voices overall.[5] For each voice, we ran a sequence of 1000 dialogue sessions, where for each session we chose a random command ("no command" was again one of the options). We added crowd-chatter noise to make the recognition task challenging. In order to be able to compare the different strategies using the same data, we pre-recorded the SAPI variables that would result from every possible action that a strategy might use. For example, suppose a random command was "go back". We would render this speech using the given voice, and then record the resulting values for the SAPI features. Then, in anticipation of some learning strategy asking for a confirmation, we rendered the appropriate speech response (e.g., "two") and recorded the resulting SAPI features. We continued this process until we had the observed SAPI features corresponding to any sequence of actions that could result from a strategy. We applied each strategy to the 1000 dialogue sessions for each voice, using the data appropriate to the actions that the strategy used. The strategies adapted the model to each voice as appropriate, and we kept a running total of the utility for each dialogue session in the sequence.

In Fig. 5, we show the average total reward for each of the different strategies, where the average is taken over the 75 voices; the error bars represent the corresponding standard errors. We performed paired t-tests to asses the significance of the differences among the algorithms. Thompson was significantly better than Interval ($t(74) = 7.17, P < 0.0001$ two-tail), Interval was significantly better than Exploit ($t(74) = 9.41, P < 0.0001$ two tail), Exploit and Boltzmann were not significantly different, Boltzmann was significantly better than Baseline ($t(74) = 4.06, P < 0.0001$ two-tail), Baseline was (not too significantly) better than Epsilon ($t(74) = 1.88$, $P = 0.06$ two-tail), and Epsilon was significantly better than DoTop ($t(74) = 2.77, P < 0.01$ two-tail).

There are some interesting observations to make from these results. First, Thompson attained significantly more utility than all other strategies, and in particular, it attained significantly more utility than the Interval strategy. This was somewhat surprising, given that Interval simply performs Thompson multiple times to determine what action to take. Evidently Interval is being too optimistic about the utility of its actions; we expect that by taking the upper bound of a narrower confidence interval, Interval will improve. A second interesting observation is the difference between Baseline and DoTop. This difference represents the added utility that we get by using the influence

[5] These test voices were a strict subset of the voices used to train the baseline model. We selected the particular voices based how natural they sounded; we wanted to simulate as closely as possible how the strategies would work for a real human voice.

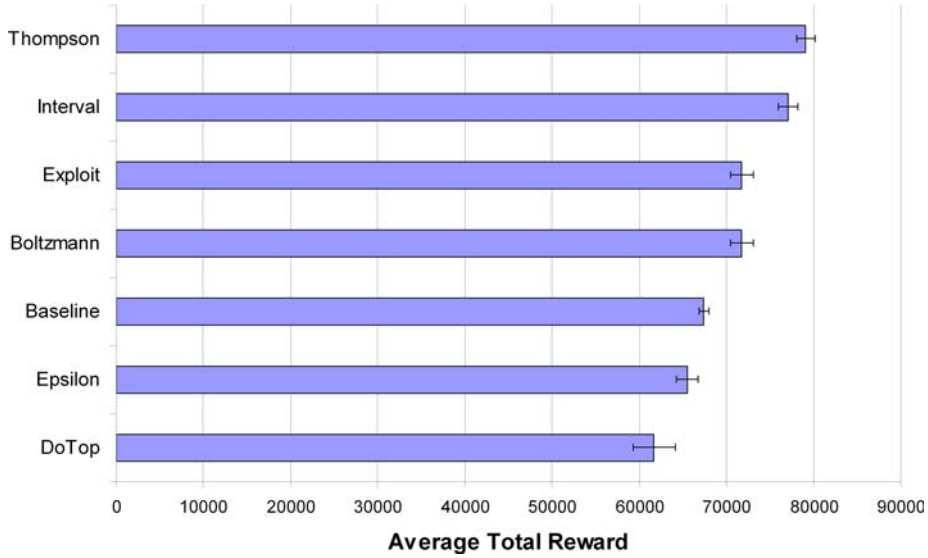

Fig. 5 Average total reward for each of the strategies

diagram to allow for repair dialogues. Finally, the difference between Thompson and Baseline represents the added utility that we get from personalizing the influence diagram to a particular voice.

To better understand the significance of the observed differences in utility, consider a simplified utility model where we either get a utility of 100 if we succeed or a utility of −100 if we fail. Under this model, the average total reward from DoTop, Baseline, and Thompson correspond to success rates of 81%, 84%, and 90%, respectively. Thus, we can interpret our results under this model as gaining 3% in our success rate as a result of using the influence diagram, and gaining an additional 6% in our success rate as a result of personalization. The *true* success rate of Baseline and Thompson are much higher than 84% and 90%, respectively, because these strategies result in successful command executions with low utilities. For example, the utility of a successful command execution after two repair actions is −50. Nonetheless, we believe the simplified utility model is useful for understanding our results.

In Fig. 6, we show a different view of our results. In particular, we plot the average, over all 75 voices, of the running average total reward as a function of the dialogue session. For example, consider the value of the Thompson curve above the kth dialogue session. For each of the 75 voices, we tracked the running average, over the first k sessions, of the total utility obtained by the Thompson strategy; what is plotted in the figure is the average, over the 75 voices, of this running average. To keep the figure simple, we omitted the corresponding error bars for each of the 1000 points. The figure shows a pronounced upward trend in all of the personalization strategies (i.e., all strategies other than Baseline and DoTop), and it suggests that the models will all continue to improve with more sessions.

In summary, our results show that Thompson was the best explore-versus-exploit strategy for personalizing influence diagrams in this domain, and that significant utility was gained by performing that personalization.

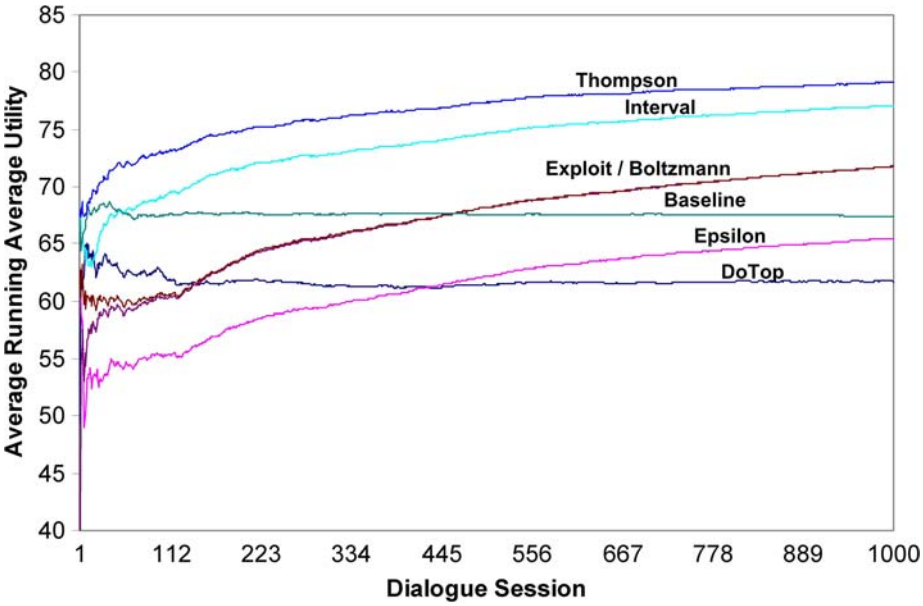

Fig. 6 Running average as a function of the iteration number

7 Conclusion

In this paper we introduced the Bandit-ID problem, which is the problem of adapting the parameters of a particular user model, the influence diagram, to a specific user. We described a number of strategies from the MDP literature, and we showed how to implement these strategies in our user-modeling scenario. We provided experimental results from a real-world user-modeling system that compare the various strategies, and our results suggest that significant differences between these strategies may occur over the long run. The Thompson strategy for personalization resulted in the highest total reward, and personalization improved total reward significantly over the non-personalized baseline model.

A natural next step for this research is to perform a user study to assess to what degree real users prefer these adaptive methods. Anecdotally, we have found that users prefer the browser instrumented with the Thompson method to one instrumented with the DoTop method. Unless the user speaks very clearly, certain commands often get misunderstood by the speech recognizer, and the adaptive methods learn quickly to engage in disambiguating confirmations. DoTop, on the other hand, will often perform the wrong action in such situations, and un-doing these actions is distracting and time consuming. In fact, we chose the confirmation utility of -75 based on our own tolerances for dialogue repairs versus the negative utility resulting from wrong actions.

There are many other interesting directions in which we can extend this work. To evaluate the various strategies, we concentrated on an empirical comparison. For a more theoretical evaluation, researchers have considered the "regret" or relative loss an agent receives for executing an exploration strategy instead of behaving optimally from the start, as determined retrospectively. Performance bounds for regret

have been developed for N-armed bandit algorithms under various assumptions (e.g., Berry and Fristedt 1985; Auer et al. 1995; Auer 2002), as well as for MAP estimation algorithms in general (Kakade and Ng 2005). It would be interesting to study the competing strategies in this framework.

Appendix

Following are all of the features that the supervised learning algorithm used to build the baseline influence diagram. The term *Rule* in a feature name refers to the user command, and the term *Token* refers to the user utterance. For example, if the user says "go back", the Rule will be BACK and the Token will be "go back".

A. Dialogue Features

1. Turn: The current dialogue step.
2. Has Confirm: Whether or not a confirmation has been performed anytime in the previous turns.
3. Number Of Repairs So Far (i): Number of repairs so far up to turn i.
4. Number Of Confirms So Far (i): Number of confirmations so far up to turn i.

B. Automatic Speech Recognition (ASR) Features (repeated for each turn i)

1. {Top|Second|Third} Rule (i): Rule that occupies the {first|second|third} position in the top-n list.
2. {Top|Second|Third} Token (i): Token that occupies the {first|second|third} position in the top-n list.
3. {Top|Second|Third} Score (i): Confidence score that occupies the {first|second|third} position in the top-n list.
4. Number of False Recognitions (i): Number of passes through SAPI's word lattice that fail to recognize a phrase.
5. Number of Interference Events (i): Number of times SAPI raised an event indicating that the recognition might have been compromised by a particular audio distortion type.
6. Most Common Interference Event (i): Most common type of audio-distortion event type raised by SAPI.
7. Number of Stream Start Events (i): Number of times an audio stream start point is detected.
8. Number of Stream End Events (i): Number of times an audio stream end point is detected.
9. Number of Stream Released (i): Number of times a stream is rejected as an audio stream.
10. Number of Sound Start Events (i): Number of times any sound start point is detected.
11. Number of Sound End Events (i): Number of times any sound end point is detected.
12. Number of Phrase Start Events (i): Number of times a phrase is detected from an audio stream.

13. Number of Audio Change Events (i): Number of times audio classification is changed.
14. Maximum Redundant {Rule|Token|Combined} (i): The cardinality of the most frequently occurring {rule|token|both}.
15. Maximum Frequency Of Redundant {Rule|Token|Combined} (i): The cardinality of distinct {rule|token|both} that repeat.
16. Score Count (i): Number of items in the n-best list
17. Score Sum (i): Sum of all the confidence scores.
18. Maximum Score (i): Maximum confidence score.
19. Minimum Score (i): Minimum confidence score
20. Score Range (i): Difference between the maximum and minimum confidence scores.
21. Score Median (i): Median confidence score if any.
22. Score Mean (i): Arithmetic mean of the confidence scores.
23. Score Geometric Mean (i): Geometric mean of the confidence scores.
24. Greatest Consecutive Difference(i): Greatest difference between any two consecutive confidence scores, if there are two or more confidence scores.
25. Score Variance (i): Variance of the confidence scores.
26. Score Stdev (i): Standard deviation of the confidence scores.
27. Score Stderr (i): Standard error of the confidence scores.
28. Score Mode (i): Mode of the confidence scores.
29. Mode Frequency (i): How frequently the mode occurs.
30. Score Skewness (i): Skewness of the distribution of confidence scores.
31. Score Kurtosis (i): Kurtosis of the distribution of confidence scores.
32. Ambiguity (i): Whether rules of the current n-best list are all the same, the top two are the same, or other.

C. Pairwise Features (defined only for $i = 2$ and $i = 3$)

1. Index of Top Rule In Previous (i): The position of the current top rule in the previous n-best list, if any.
2. Index of Top Rule In First Slice(i): The position of the current top rule in the first-turn n-best list, if any.
3. Index of Top Token In Previous (i): The position of the current top token in the previous n-best list, if any.
4. Index of Top Token In First Slice (i): The position of the current top token in the first turn n-best list, if any.
5. Score More Than Previous (i): Whether the average confidence score is greater than the previous average confidence score.
6. Gap Between Top Scores (i): Difference between the current top confidence score and the previous top confidence score.
7. Gap Between Top Scores With First Slice (i): Difference between the current top confidence score and the first-turn top confidence score.

References

Albrecht, D., Zukerman, I., Nicholson, A.: Bayesian models for keyhole plan recognition in an adventure game. User Model. User-Adapted Interaction, Special Issue Machine Learning User Model. **8**(1–2), 5–47 (1998)

Auer, P.: Using confidence bounds for exploitation-exploration trade-offs. J. Machine Learn. Res. **3**, 397–422 (2002)
Auer, P., Cesa-Bianchi, M., Freund, Y., Schapire, R.: Gambling in a rigged casino: the adversarial multi-armed bandit problem. In: In Proceedings of the 36th Annual Symposium on Foundations of Computer Science, pp. 322–331. IEEE Computer Society Press, Los Alamitos, CA (1995)
Berry, D., Fristedt, B.: Bandit Problems: Sequential Allocation of Experiments Chapman and Hall, London (1985)
Boutilier, C., Dean, T., Hanks, S.: Decision-theoretic planning: structural assumptions and computational leverage. J. Aritif. Intell. Res. **1**, 1–93 (1999)
Chickering, D.M.: The winmine toolkit. Technical Report MSR-TR-2002-103, Microsoft Redmond, WA (2002)
Cooper, G.F.: A method for using belief networks as influence diagrams. In: Heckerman, D., Mamdani, A., (eds.), Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence, Washington, DC, pp. 55–63. Morgan Kaufmann (1993)
Dearden, R., Friedman, N., Russell, S.: Bayesian Q-learning. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence, pp. 761–768. Madison, WI (1998)
Heckerman, D.: A Bayesian approach for learning causal networks. In: Hanks, S., Besnard, P. (eds.), Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, Montreal, QU. Morgan Kaufmann (1995)
Heckerman, D.: A tutorial on learning Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research (1996)
Horvitz, E., Breese, J., Heckerman, D., Hovel, D., Rommelse, K.: The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, pp. 256–265. Madison, Wisconsin (1998)
Howard, R., Matheson, J.: Influence diagrams. In: Readings on the Principles and Applications of Decision Analysis, Vol. II, pp. 721–762. Strategic Decisions Group, Menlo Park, CA (1981)
Kaelbling, L.P.: Learning in Embedded Systems. The MIT Press, Cambridge, MA (1993)
Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. J. Artif. Intell. Res. **4**, 237–285 (1996)
Kakade, S.M., Ng, A.Y.: Online bounds for bayesian algorithms. In: Saul, L.K., Weiss, Y., Bottou, L. (eds.), Advances in Neural Information Processing Systems, Vol. 17, pp. 641–648. MIT Press, Cambridge, MA (2005)
Lauritzen, S.L., Nilsson, D.: Representing and solving decision problems with limited information. Manage. Sci. **47**(9), 1235–1251 (2001)
Roy, N., Pineau, J., Thrun, S.: Spoken dialogue management using probabilistic reasoning. In: Proceedings of ACL-2000, pp. 93–100. Hong Kong, China (2000)
Shachter, R., Peot, M.: Decision making using probabilistic inference methods. In: Proceedings of the 8th Annual Conference on Uncertainty in Artificial Intelligence, pp. 276–283. San Mateo, CA, Morgan Kaufmann Publishers (1992)
Singh, S., Litman, D., Kearns, M., Walker, M.: Optimizing dialogue management with reinforcement learning: experiments with the njfun system. J. Artif. Intell. Res. **16**, 105–133 (2002)
Sutton, R., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
Tatman, J.A., Shachter, R.D.: Dynamic programming and influence diagrams. IEEE Trans. Syst. Man Cybernet. **20**(2), 365–379 (1990)
Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometricka. **25**, 285–294 (1993)
Wyatt, J.: Exploration and Inference in Learning from Reinforcement. PhD thesis, University of Edinburgh (1997)
Young, S.: Probabilistic methods in spoken dialogue systems. Philos. Trans. Roy. Soc. (Ser A) **358**(1769), 1389–1402 (2000)
Zukerman, I., Albrecht, D.: Predictive statistical models for user modeling. User Model. User-Adapted Interact. **11**(1), 5–18 (2001)

Authors' vitae

Max Chickering has been working in the Machine Learning and Applied Statistics group of Microsoft Research and is now with Microsoft Live Labs. He is interested in practical applications of machine learning algorithms. Max received his PhD in computer science from the University of California at Los Angeles, and he received his Bachelors in computer science from the University of California at Berkeley.

Time Paek is a researcher in the Machine Learning and Applied Statistics group at Microsoft Research. Tim received his M.S. in Statistics and Ph.D. in Cognitive Psychology from Stanford University, and his B.A. in Philosophy from the University of Chicago. His primary research focus is on spoken dialogue systems. With a keen interest in enhancing deployed systems, he has pursued research in the following areas: dialogue management, user modeling, personalization, machine learning and human–computer interaction.