A POMDP Formulation of Preference Elicitation Problems

Craig Boutilier
Department of Computer Science
University of Toronto
Toronto, ON, M5S 3H5, CANADA
cebly@cs.toronto.edu

Abstract

Preference elicitation is a key problem facing the deployment of intelligent systems that make or recommend decisions on the behalf of users. Since not all aspects of a utility function have the same impact on object-level decision quality, determining which information to extract from a user is itself a sequential decision problem, balancing the amount of elicitation effort and time with decision quality. We formulate this problem as a partially-observable Markov decision process (POMDP). Because of the continuous nature of the state and action spaces of this POMDP, standard techniques cannot be used to solve it. We describe methods that exploit the special structure of preference elicitation to deal with parameterized belief states over the continuous state space, and gradient techniques for optimizing parameterized actions. These methods can be used with a number of different belief state representations, including mixture models.

1 Introduction

Preference elicitation (PE) is a fundamental problem in the development of intelligent decision tools and autonomous agents. Software and agents of this type are often charged with the task of making decisions, or recommending courses of action, for a specific user. Making optimal decisions on behalf of a user requires knowing some information about her preferences or utility function. It is important to keep in mind that utility functions can vary widely from user to user (even while the other ingredients of a decision scenario, such as system dynamics, remain fixed across users). For this reason, preference elicitation—the process of extracting the necessary preference or utility information from a user—is arguably one of the more important problems facing AI. Applications of elicitation processes are pervasive, ranging from low-stakes decision processes (e.g., the control of user interaction with a product Web site) to critical decision assessment systems (e.g., clinical decision making [5; 6]).

The elicitation of preference and utility functions is complicated by the fact that utility functions are very difficult for users to assess [8; 11]. The burden of elicitation can be lessened considerably in a given *specific* decision scenario (or a restricted set of scenarios). An optimal decision can usually be made without full knowledge of a user's preferences. For example, if some outcomes (or attribute values) simply aren't possible in a specific situation, the utilities for those outcomes (or values) have no bearing on the decision problem. Furthermore, even "relevant" utility information may have only marginal impact on decision quality. If the cost of obtaining that information exceeds the benefit it provides, then this information too can be safely ignored.

The development of optimal *elicitation strategies*, that address the tradeoff between effort expended in elicitation and the impact on object-level decision quality, has been explored very little. The chief exception is the work of Chajewska, Koller, and Parr [6], who model this problem by assuming a distribution over utility functions, and refining uncertainty over a user's utility as queries are answered. Their myopically optimal elicitation strategy involves asking the (single) query with greatest *expected value of information* with respect to the current distribution.

In this paper, we extend this point of view in several ways, the most important of which is viewing elicitation as a *sequential* decision problem to be solved (approximately) optimally rather than myopically. Intuitively, a myopic approach can fail to ask the correct questions because it neglects the value of *future* questions when determining the value of the *current* question. As is well known (and as we demonstrate), greedy approaches to computing value of information can underestimate the value of information if value can only be obtained from a sequence of queries. Specifically, if no single question can cause a change in the optimal decision, a myopic approach will never try to reduce its uncertainty. The model of [6] can easily be extended to do multistage lookahead, overcoming this difficulty. However, since the required computations in this model are online, we instead pose the elicitation problem as a partially-observable Markov decision process (POMDP). The POMDP perspective also allows for more suitable termination criteria, noisy response models, and permits policies to be constructed for arbitrary belief states (rather than solving the problem online for a fixed prior). In addition, optimal value functions (and implicitly, optimal policies) can be computed offline, allowing for fast online response during the elicitation process itself.

Conceptually, the POMDP formulation is straightforward; but practically, difficulties emerge because of the continuous nature of the underlying state (utility function) and action (query) spaces. We propose several methods for dealing with these problems that exploit the structure of the elicitation problem. In particular, we propose an approach to approximating the optimal value function that handles the continuous action and state spaces of the POMDP effectively, and allows for the concise representation of value functions for belief states represented using mixture models.

2 The Underlying Decision Problem

We assume that we have a system charged with making a decision on behalf of a user in a specific *decision scenario*. By a decision scenario, we refer to a setting in which a fixed set of choices (e.g., actions, policies, recommendations) are available to the system, and the (possibly stochastic) effects of these choices are known. The system's task is to take the decision with maximum expected utility with respect to the user's utility function over outcomes, or some approximation thereof. The system may have little information about the user's utility function, so to achieve this aim it must find out enough information to enable a good decision to be made. We assume that the system has available to it a set of queries it can ask of the user that provide such information.[1]

Formally, assume a *decision scenario* consists of a finite set of possible *decisions* D, a finite set of n possible outcomes (or states) S, and a distribution function $\Pr_d \in \Delta(S)$, for each $d \in D$.[2] The term $\Pr_d(s)$ denotes the probability of outcome s being realized if the system takes decision d. A utility function $u : S \rightarrow [0, 1]$ associates utility $u(s)$ with each outcome s. We often view u as a n-dimensional vector $\boldsymbol{u}$ whose ith component u_i is simply $u(s_i)$. We assume that utilities are normalized in the range $[0, 1]$ for convenience. The *expected utility* of decision d with respect to utility function $\boldsymbol{u}$ is:

$$EU(d, \boldsymbol{u}) = \boldsymbol{p}_d \boldsymbol{u} = \sum_{i \in S} \Pr_d(s_i) u_i$$

Note that $EU(d, \boldsymbol{u})$ is linear in $\boldsymbol{u}$. The optimal decision d^* w.r.t. $\boldsymbol{u}$ is that with *maximum expected utility (MEU)*.[3]

In general, the utility function $\boldsymbol{u}$ will not be known with certainty at the start of the elicitation process (nor at its end). Following [6], we model this uncertainty using a density P over the set of possibly utility functions

$$U = \{\boldsymbol{u} : \boldsymbol{0} \leq \boldsymbol{u} \leq \boldsymbol{1}\} = [0, 1]^n$$

If a system makes a decision d under such conditions of uncertainty, the expected utility of d must reflect this. We define the expected utility of d given density P over U as follows:

$$EU(d, P) = \int \boldsymbol{p}_d \boldsymbol{u} P(\boldsymbol{u}) d\boldsymbol{u}$$

Since EU is linear is $\boldsymbol{u}$, $EU(d, P)$ can be computed easily if the expectation of $\boldsymbol{u}$ w.r.t. P is known. In such a state of uncertainty, the optimal decision is that d^* with maximum expected utility $EU(d^*, P)$. We denote by $MEU(P)$ the value of being in state of P, assuming one is forced to make a decision: $MEU(P) = EU(d^*, P)$.[4]

In order to reduce its uncertainty about the user's utility function, the system has available to it a set of *queries* Q. With each query q is associated a finite set of possible *responses* $R_q = \{r_q^1, \cdots, r_q^m\}$. A common type of query is the *standard gamble w.r.t. outcome* s_i, where the user is asked if she prefers s_i to a gamble in which the best outcome $s_\top$ occurs with probability l and the worst $s_\bot$ occurs with probability $1 - l$ [11]. Note that $u(s_\top) = 1$ and $u(s_\bot) = 0$ given our normalization assumptions. We designate this query $q_i(l)$ and focus our attention on standard gamble queries. Of course, many other query types can be captured within our generic model.

For standard gamble queries we have a binary response space: $R(q) = \{yes, no\}$. The responses to a query may be noisy: we assume a *response model* of the form $\Pr(r_q^i | q, \boldsymbol{u})$ which denotes the probability of any response $r_q^i \in R_q$ to question q by a user with true utility $\boldsymbol{u}$. To keep the presentation simple, we assume fixed false positive and false negative probabilities for each query type q_i: so $\Pr(yes | q_i(l), u_i < l) = p_{fp}^i$, while $\Pr(no | q_i(l), u_i \geq l) = p_{fn}^i$. We let $p_{tp}^i = 1 - p_{fn}^i$ and $p_{tn}^i = 1 - p_{fp}^i$ be the probabilities of "correct" positive and negative responses.

Finally, each question q has a cost $c(q)$. This reflects the difficulty the user is expected to have in answering the question due to the mental burden in imposes on the user, the computational costs associated with computing an answer, the time required to process the question, or many other factors.[5]

Given a response r to a question q, the updated conditional density P_r can be determined by application of Bayes rule:

$$P_r(\boldsymbol{u}) = P(\boldsymbol{u}|r) = \frac{\Pr(r|q, \boldsymbol{u})P(\boldsymbol{u})}{\int \Pr(r|q, \boldsymbol{u})P(\boldsymbol{u})d\boldsymbol{u}} \quad (1)$$

The (myopic) *expected value of information (EVOI)* of a query can be defined by considering the difference between $MEU(P)$ and the expectation (w.r.t. r) of $MEU(P_r)$. A query can be deemed worthwhile if its EVOI outweighs its cost, and a myopically optimal elicitation strategy involves asking queries with maximal EVOI at each point [6].

3 Preference Elicitation as a POMDP

Value of information plays an important role in good elicitation strategies, as proposed in [6]. We take a different, though similarly motivated, approach by formulating the elicitation problem as a POMDP. This view makes the sequential nature of the elicitation problem clear and avoids problems facing myopic EVOI. Furthermore, by posing the problem as a

---

[1]We use the term "queries" for concreteness; any interaction with a user can provide (noisy) information about her utility function.

[2]The extension of our elicitation methods to a set of possible decision scenarios is straightforward.

[3]If u is represented using some more concise model, such as a linear utility model [11] or a graphical model [4], $\boldsymbol{u}$ is simply the vector of parameters required for that model. All results below apply.

[4]Taking the expectation w.r.t. $P(U)$ is not unproblematic: certain "calibration assumptions" of the elements of U is necessary to ensure that this expectation makes sense [3]. We do not discuss this subtlety further, but the disquieted reader can treat U as expected monetary value or some other surrogate.

[5]To keep the model simple, we assume questions have a constant cost, though costs depending on $\boldsymbol{u}$ are often reasonable.

POMDP we have access to solution techniques for constructing a policy that covers all possible initial belief states (as opposed to one policy designed for a fixed prior as in [6]). Of course, many computational difficulties must be overcome (see the next section); but if these can be surmounted, the advantages of a full POMDP formulation are clear.

3.1 A POMDP Formulation

The POMDP formulation is quite direct. The set of system states is U, the set of possibly utility functions. The set of belief states of the POMDP is simply the set of densities over U. Since our state space is a n-dimensional continuous space, we will require some parameterized representation of belief states. The system dynamics are trivial: the underlying utility function $\boldsymbol{u}$ never changes, so at each time t, $\boldsymbol{u}$ is exactly as it was at time $t-1$. The actions available to the system are queries Q and decisions D; we let $A = Q \cup D$. Queries induce no change in the underlying system state $\boldsymbol{u}$, but do provide information about it. Each decision d is a terminal action. The cost $c(q)$ of question q is state-independent; but the terminal reward associated with a decision d does depend on the state: $Rew(d, \boldsymbol{u}) = EU(d, \boldsymbol{u})$. The sensor model for the POMDP is the response model $\Pr(r_q|q, \boldsymbol{u})$. Assuming standard gamble queries, we have a continuous action space: for each outcome s_i, we have queries of the form $q_i(l)$ for any $l \in [0, 1]$. We assume an infinite horizon model (since the process terminates at an unspecified time) with discount factor γ.

We can formulate the optimal value function and policy using the standard Bellman equations over the fully-observable *belief state MDP* [14]. We define the optimal value function V^*, ranging over belief states P, as:

$$V^*(P) = \max_{a \in A} Q_a^*(P)$$

where the Q-functions Q_a^* are defined for each query and decision. For decisions d, we have $Q_d^*(P) = EU(d, P)$. For queries $q_i(l)$ we parameterize Q_i^* by the lottery probability l:

$$Q_i^*(l, P) = c(q_i(l)) + \gamma \sum_{r \in R} \Pr(r|q_i(l), P) V^*(P_r)$$

Finally, the optimal (stationary) policy π^* is given by the action, decision or query, maximizing the value function.

3.2 Belief State Representation

Because the underlying state space is a multidimensional continuous space, solving a POMDP of this type, as well as performing belief state maintenance for policy implementation, requires a reasonable belief state representation. In such circumstances, some parametric form is often used; for instance, an n-dimensional Gaussian might be used to represent belief states (perhaps truncated at the utility boundaries). Particle filter models can also be used to solve continuous state POMDPs [15].

One difficulty with parametric models is their inflexibility. A mixture of Gaussians offers considerably more flexibility in this regard—this representation is adopted in [6]. A difficulty with Gaussian mixtures is that they are not closed under
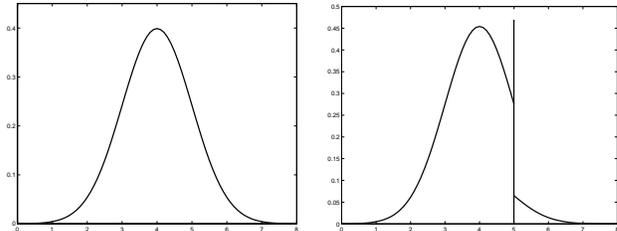

Figure 1: Update of Gaussian after query $q_i(0.5)$.

belief state update using standard gamble queries.[6] Specifically, under our model, if P is Gaussian, then P_r is a mixture of two *truncated* Gaussians, as shown in Figure 1. For instance, given the answer *yes* to query $q_i(l)$, we have $P_r(\boldsymbol{u}) = \alpha^- P(\boldsymbol{u})$ if $u_i < l$ and $P_r(\boldsymbol{u}) = \alpha^+ P(\boldsymbol{u})$ if $u_i \geq l$, where $\alpha = p_{tp}^i \int_{u_i < l} P(\boldsymbol{u}) + p_{fp}^i \int_{u_i \geq l} P(\boldsymbol{u})$ is a normalizing constant, $\alpha^- = p_{tp}^i/\alpha$ and $\alpha^+ = p_{fp}^i/\alpha$. The relative weights of the two components is given by, $w^- = \alpha^- \int_{u_i < l} P(\boldsymbol{u})$ and $w^+ = \alpha^+ \int_{u_i \geq l} P(\boldsymbol{u})$.

While Gaussian mixtures aren't closed under update, after conditioning on a response, this "truncated" mixture model can be sampled and refit using a standard method such as EM [2]. This technique is adopted in [6]. An alternative we consider here is to use mixtures of truncated Gaussians directly as a belief state representation. This has two advantages. First, one needn't refit the belief state using computationally expensive procedures like EM—given an initial mixture, we simply retain the truncated mixtures that result from updating after a series of queries. Second, because the truncated mixture is exact with respect to the initial belief state, no belief state approximation is needed. A drawback is that the number of truncated components doubles with each update. A suitable compromise involves maintaining a truncated mixture until the number of components becomes unwieldy, then refitting this model to a new mixture using EM.

We also consider the use of uniform distributions, a parametric form that meshes well with the queries at hand. We use a prior consisting of a mixture of k uniform distributions. Updating a uniform after query $q_i(l)$ results in a mixture of two uniforms identical to the original except with the upper and lower bounds in dimension i revised. Again, EM can be used to reduce the number of components periodically.

3.3 Difficulties Facing the Formulation

Even with good belief representation, standard methods for solving a POMDP cannot be used. Methods for finite spaces rely on the fact that the optimal value function is (approximately) piecewise linear and convex (PWLC) [14]. Continuous state problems require some form of special structure

---

[6]For certain forms of queries, Kalman filter-like techniques can be used for updating Gaussians however.

(such as Gaussian belief states, linear dynamics, etc.) or the use of function approximation. The value function for the elicitation POMDP has no convenient closed form, so function approximation is needed. Let our belief state have a parametric form with parameter vector $\boldsymbol{\theta}$ (e.g., the weights of a Gaussian mixture together with the mean and covariance parameters of each component). Our goal is then to construct an estimate $\widetilde{V}(\boldsymbol{\theta})$ of the optimal value function.

Several general approaches to value function approximation for POMDPs have been proposed. Those motivated by the PWLC nature of the value function [7; 13] are not appropriate here. More general approximators, such as feedforward neural networks, seem more promising [12]. Grid-based models [10] can also used, but require computationally demanding projections (refitting) of updated belief states for each DP update. Policy search methods [9] and clustering to discretize utility space [5] might also be used.

We'll also require a method for dealing with the continuous action space. Again, unless there is some special structure, special techniques will be required to handle the large action space. Fortunately, our action space is well-parameterized; specifically, we have a finite number of query types q_i, with each type parameterized by a lottery probability l.

4 An Approximation Technique for PE

To deal with the difficulties described above we propose a technique for solving the PE POMDP using a suitable function approximation architecture. We assume our belief state is captured by a mixture model with an unspecified number of components (recall that generally the number of components increases over time). Our aim will be to compute the value function only for a single parametric component rather than for the entire mixture. Online policy implementation will require that we use this "single component" value function to determine the value of a mixture density.

4.1 Approximating the Value Function

We use $\boldsymbol{\theta}$ to denote the parameter vector for a single belief state component. Rather than approximating the value function directly, we will compute Q-functions, one for each query type q_i. The approximator $Q_i(l, \boldsymbol{\theta})$ is parameterized by the lottery probability l. We don't approximate Q_d for decisions d, since their exact computation is straightforward. We assume an approximation architecture with parameters or *weights* $\mathbf{w}$, that can be trained in some standard fashion. We also assume that the Q_i are differentiable with respect to l.[7]

We solve the PE POMDP using a form of asynchronous approximate value iteration [1]. At each iteration we sample a (non-mixture) belief state $\boldsymbol{\theta}$ and a query $q_i(l)$ according to some sampling schedule. We compute the backed up Q-value of query $q_i(l)$ at state $\boldsymbol{\theta}$ using the current approximations:

$$Q_i(\boldsymbol{\theta}, l) = c(q_i) + \gamma[\Pr(yes|q_i(l), \boldsymbol{\theta})V(\boldsymbol{\theta}_{yes}) + \Pr(no|q_i(l), \boldsymbol{\theta})V(\boldsymbol{\theta}_{no})] \quad (2)$$

(here $\boldsymbol{\theta}_{yes}$ denotes the updated belief state given response yes, and similarly for no). Letting q_{new} denote the backed up

---
[7]We discuss approximation architectures below.

value, we then train our approximator Q_i with inputs l and $\boldsymbol{\theta}$ and output q_{new}.

Unfortunately, the computation of Eq. 2 is not straightforward. First, the resulting densities are mixtures of the underlying components. For instance, if $\boldsymbol{\theta}$ is a truncated Gaussian, then $\boldsymbol{\theta}_{yes}$ is a mixture of two truncated Gaussians $\boldsymbol{\theta}_{yes}^1$ and $\boldsymbol{\theta}_{yes}^2$, with mixing weights α_1 and α_2. Unfortunately, the value function (via the Q-functions) is only defined for single components. Furthermore, determining $V(\boldsymbol{\theta}_{yes})$ from the Q-function approximators requires the solution of:

$$V(\boldsymbol{\theta}_{yes}) = \max[\max_d Q_d(\boldsymbol{\theta}_{yes}), \max_i \max_l Q_i(\boldsymbol{\theta}_{yes}, l)] \quad (3)$$

Maximization over decisions d and query types q_i is straightforward, but maximizing over the continuous input l is less so. (Similar remarks obviously apply to $V(\boldsymbol{\theta}_{no})$.)

We deal with the first problem by taking advantage of the fact that we construct Q-functions—rather than the value function directly—to determine Q-values of the necessary mixtures. Specifically, suppose we have belief state $\boldsymbol{\theta} = \alpha_1 \boldsymbol{\theta}_1 + \alpha_2 \boldsymbol{\theta}_2$ that is the mixture of two components. Then the expected value of a decision d for the mixture is:

$$\begin{aligned} Q_d(\boldsymbol{\theta}) &= Q_d(\alpha_1 \boldsymbol{\theta}_1 + \alpha_2 \boldsymbol{\theta}_2) \\ &= \alpha_1 Q_d(\boldsymbol{\theta}_1) + \alpha_2 Q_d(\boldsymbol{\theta}_2) \quad (4) \end{aligned}$$

A similar fact holds for queries:

Proposition 1 $Q_i(\alpha_1 \boldsymbol{\theta}_1 + \alpha_2 \boldsymbol{\theta}_2, l) = \alpha_1 Q_i(\boldsymbol{\theta}_1, l) + \alpha_2 Q_i(\boldsymbol{\theta}_2, l)$

Proof: Let $\boldsymbol{\theta}$ denote the mixture $\alpha_1 \boldsymbol{\theta}_1 + \alpha_2 \boldsymbol{\theta}_2$. We observe that the update of $\boldsymbol{\theta}$ given response yes to query $q_i(l)$ is a mixture $\boldsymbol{\theta}^{yes} = \alpha_1^{yes} \boldsymbol{\theta}_1^{yes} + \alpha_2^{yes} \boldsymbol{\theta}_2^{yes}$, where

$$\alpha_1^{yes} = \frac{\alpha_1 \Pr(yes|\boldsymbol{\theta}_1)}{\Pr(yes|\boldsymbol{\theta})}, \quad \alpha_2^{yes} = \frac{\alpha_2 \Pr(yes|\boldsymbol{\theta}_2)}{\Pr(yes|\boldsymbol{\theta})} \quad (5)$$

and $\boldsymbol{\theta}_i^{yes}$ is the update of the component $\boldsymbol{\theta}_i$ given a yes response. Similar expressions hold for $\boldsymbol{\theta}^{no}$.

Letting c denote the query cost and suppressing the query $q_i(l)$ on the righthand side of the conditioning bars, we have:

$$\begin{aligned} Q_i(\boldsymbol{\theta}, l) &= c + \gamma[\Pr(yes|\boldsymbol{\theta})V(\boldsymbol{\theta}^{yes}) + \Pr(no|\boldsymbol{\theta})V(\boldsymbol{\theta}^{no})] \\ &= c + \gamma[\alpha_1^{yes} \Pr(yes|\boldsymbol{\theta})V(\boldsymbol{\theta}_1^{yes}) + \alpha_2^{yes} \Pr(yes|\boldsymbol{\theta})V(\boldsymbol{\theta}_2^{yes}) \\ &\quad + \alpha_1^{no} \Pr(no|\boldsymbol{\theta})V(\boldsymbol{\theta}_1^{no}) + \alpha_2^{no} \Pr(no|\boldsymbol{\theta})V(\boldsymbol{\theta}_2^{no})] \\ &= c + \gamma[\alpha_1 \Pr(yes|\boldsymbol{\theta}_1)V(\boldsymbol{\theta}_1^{yes}) + \alpha_2 \Pr(yes|\boldsymbol{\theta}_2)V(\boldsymbol{\theta}_2^{yes}) \\ &\quad + \alpha_1 \Pr(no|\boldsymbol{\theta}_1)V(\boldsymbol{\theta}_1^{no}) + \alpha_2 \Pr(no|\boldsymbol{\theta}_2)V(\boldsymbol{\theta}_2^{no})] \\ &= c + \gamma[\alpha_1(\Pr(yes|\boldsymbol{\theta}_1)V(\boldsymbol{\theta}_1^{yes}) + \Pr(no|\boldsymbol{\theta}_1)V(\boldsymbol{\theta}_1^{no})) \\ &\quad + \alpha_2(\Pr(yes|\boldsymbol{\theta}_2)V(\boldsymbol{\theta}_2^{yes}) + \Pr(no|\boldsymbol{\theta}_2)V(\boldsymbol{\theta}_2^{no}))] \\ &= \alpha_1 Q_i(\boldsymbol{\theta}_1, l) + \alpha_2 Q_i(\boldsymbol{\theta}_2, l) \end{aligned}$$

The third equality is obtained using the expressions in Eq. 5 (and analogous expressions for the no response). ◄

We deal with the second issue, maximization, as follows. Maximization over the decisions $d \in D$ is straightforward. For a given query q_i, the maximization over lottery probabilities l is achieved using some suitable optimization technique. For certain function approximators, this optimization may be computed analytically. Otherwise, since the approximator is

differentiable with respect to input l (which is true for most common approximators), we can—given a fixed single component belief state—approximate the value of $\max_l Q_i(\boldsymbol{\theta}, l)$ using gradient ascent. Specifically, since $\frac{\partial Q_i(\boldsymbol{\theta}, l)}{\partial l}$ is defined, we can find a local maximum readily.[8] Of course, $\boldsymbol{\theta}_{yes}$ is a mixture of two truncated Gaussians. But we can still use gradient ascent by noting that:

$$\frac{\partial Q_i(\boldsymbol{\theta}_{yes}, l)}{\partial l} = \alpha_1 \frac{\partial Q_i(\boldsymbol{\theta}^1_{yes}, l)}{\partial l} + \alpha_2 \frac{\partial Q_i(\boldsymbol{\theta}^2_{yes}, l)}{\partial l} \quad (6)$$

Consequently, $V(\boldsymbol{\theta}_{yes})$ can be determined by computing the value of each decision d, using Eq. 6 to determine the maximum value of a query of the form q_i for each i, and then maximizing over these $|S \cup D|$ values. The value given a *no* response is determined analogously. These are combined using Eq. 2 to get an backed up estimate of $Q_i(\boldsymbol{\theta}, l)$.

4.2 Online Policy Execution

The procedure above produces a collection of Q-functions, one for each query type q_i. With this in hand, we can determine the optimal action for a single component belief state $\boldsymbol{\theta}$ as follows: compute $Q_d(\boldsymbol{\theta})$ for each d, compute $\max_l Q_i(\boldsymbol{\theta}, l)$ for each q_i using gradient ascent, then choose the action, d or $q_i(l)$, with maximum value. The resulting belief state is, however, a mixture. But even though we have computed Q-functions explicitly only for single components, we can use these to directly determine the optimal action for a mixture using the ideas described above. Optimizing over l for a mixture of k components can be accomplished using the same technique as optimizing for a 2-mixture (using the obvious extension of Eq. 6). We describe practical enhancements to this scheme in the next section.

Online policy implementation then consists of two basic steps per stage: (a) if belief state $b^t = \{\boldsymbol{\theta}^t[i], w^t[i]\}$ denotes the mixture belief state at stage t, we choose the optimal action for b, and execute it (either asking a query or recommending a decision); (b) if the action is a query, we update our belief state w.r.t. the response to produce b^{t+1}, and continue.

One difficulty with this online policy implementation technique lies in the fact that the number of mixture components may quickly become unmanageable. If the prior is a single component mixture, we will have (up to) 2^t components after t queries. Note that this fragmentation is purely a function of belief state update, and not due to policy implementation *per se*. But belief state maintenance is much more tractable if the number of components is reduced periodically. One way to do this is to prune components with low weight. This is computationally feasible, but must be done with some care to ensure that important parts of "total belief space" are not ignored. An alternative is to simply fit the current belief state to a smaller mixture model using EM. This is more computationally demanding, but will provide more accurate results if dramatic reduction in model size is desired. Computation is a critical consideration since refitting needs to be done online; but periodic refitting can be managed intelligently.

---

[8]We can show that the optimal Q-function for query i is concave in l, thus if our approximator reflects this fact, we can be reasonably confident that a global maximum will be found.
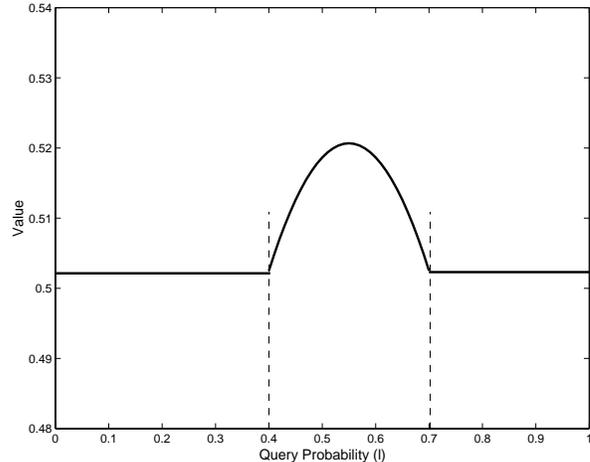

Figure 2: A 1-D View of a 2-stage Q-function.

5 Practical Tricks and Empirical Results

In this section we describe some preliminary empirical results using the elicitation method described. In our experiments we use mixtures of uniform distributions as our belief state model since updating and computing the required expectations is quite simple (as the model fits very well with the form of the queries we use).

5.1 Practical Enhancements

Each Q-function approximator has as input the parameters of an n-dimensional uniform (i.e., upper and lower bounds in each dimension) as well as a query point l. All of our results use a quadratic function approximator. We have experimented with linear approximators and feedforward neural networks as well. Linear approximators clearly do quite poorly, as the Q-functions are obviously nonlinear. Figure 2 illustrates a 1-D slice of the optimal two-stage-to-go Q-function at a fixed belief state, which is uniform (in the dimension shown) on interval $[0.4, 0.7]$. The Q-function is concave in l and quadratic in the difference of l and the mean (over the region of positive density).[9] Quadratic approximators seem to provide good quality solutions over the range of the belief state. Given an n-dimensional belief space, the quadratic approximator has $2n + 1$ inputs, and $O(n^2)$ weights to be tuned. Tuning was accomplished using simple gradient descent in weight space with momentum. Optimization over lottery probabilities l (when computing Bellman backups) was accomplished using no specific gradient information (instead Matlab's FMNBND function was used). The training schedule we used was as follows: we first trained the approximators with 2-stage-to-go Q-functions. The reason for this was that these backups require no "bootstrapping" since the one-stage function can be computed exactly (since it involves only decisions not queries). This helped quickly capture the quadratic dependence of Q-functions on the difference between the input l and the mean

---

[9]In general, the optimal t-stage Q-function is a degree $2t$ polynomial for a uniform distribution.

vector. After a fixed number of 2-stage backups at systematically chosen belief points, we generated random belief states and queries and applied the backup procedure described in the previous section.[10]

We exploit certain properties of value of information to restrict training and overcome approximator deficiencies. For example, given a density with upper and lower bounds u_i and l_i in dimension i, a query $q_i(c)$ with c outside that range has no value (i.e., will not alter the belief state). We restrict training to queries that lie within the range of the belief state. Since approximators generalize, they tend to assign value to queries outside the range that are *lower* than their true value, due to the nature of VOI (see Figure 2). For this reason, we restrict the value of a query outside the range of the belief state to be equal to the value at its boundary. Although asking queries outside the range of a single mixture component needn't be considered in unmixed models, assigning values to such queries is necessary when computing VOI w.r.t. a *mixture*.

Online policy implementation reflects the considerations described above. Our current implementation uses some simple pruning strategies to keep the number of mixture components manageable (pruning components of low weight), but no sophisticated refitting of mixtures. Because of the crudeness of the Q-function approximators, some heuristics are also used to ensure progress is made. For example, we only ask queries such that the mass of the belief state on either side of the query point is greater than some threshold δ.

5.2 Empirical Results

Our first example is reasonably small, involving a decision problem with four outcomes, with six decisions. For each outcome, there is one decision that achieves that outcome with probability 0.7 (and each other outcome with probability 0.1). The other two decisions achieve two of the outcomes with moderate probability 0.4, and the remaining two decision with low probability (0.1). Each query has a cost of 0.08 and noise parameter (i.e., false positive and false negative rates) 0.03. Recall that utilities lie in the range $[0, 1]$.

Figure 3 shows a plot of the sampled Bellman error as a function of the number of random backups, starting immediately after the initial systematic sweep of belief space (64000 backups). Bellman error is determined by sampling 2000 random belief-state query pairs and evaluating Bellman error at each. Results are averaged over 10 runs (error bars show standard deviation w.r.t. these runs). Average error is on the order of 2.5 per cent. Computation time for the initial backups is minimal, since 2-stage backups require no bootstrapping (hence no optimization over query points): on average these 64000 these took 83s. For the other backups, computation time is more intensive: the 160000 backups shown take on average 11899s (about 3.3 hours), or .074s per backup. The 70-fold increase in time per backup is almost exclusively due to the optimization of the query probabilities when assessing the max Q-value of a query type. This is largely due to the fact that our preliminary implementation relies on a generic optimization procedure. We expect much better performance
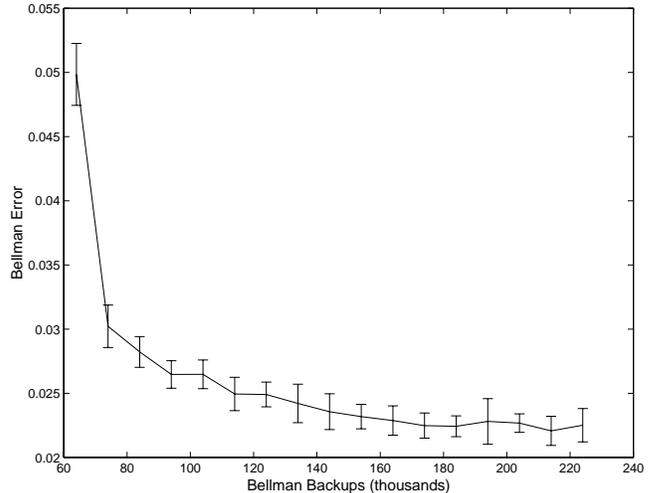

Figure 3: Bellman Error (over 10 runs): Four outcomes.

when gradient information is used to do optimization. We note, however, that: (a) the expense of this computation is borne offline; and (b) this level of Bellman error is approached long before all backups are completed.

The value of offline value function construction is borne out: determining the optimal action online takes minimal time (several hundredths of a second) and scales linearly with the number of mixture components. We report on a systematic evaluation of the number of queries in a longer version of the paper, including a comparison to the greedy query strategy . Generally, the implementation asks few queries; but we do notice that frequently the procedure will become stuck in loops or ask nonsensical queries due to the looseness of the approximate Q-functions. We are currently exploring more refined approximators, including feedforward NNs, and piecewise linear approximators.

We also ran an example with 20 outcomes and 30 decisions. Computation time per Bellman backup increases substantially because of the need to optimize over 20 queries at each backup (and the increase in the number of approximator parameters). Average backup time for 139000 2-stage backups is 0.002s, while for the 240000 full backups shown is 0.57s. The quadratic approximator does a reasonable job of capturing the value function: average Bellman error for one sample run is illustrated in Figure 4.

To illustrate the difficulty with the myopic approach, we compare the myopic strategy of [6] to the sequential model developed here on the following simple example. We have seven outcomes $s_1, \ldots, s_7$, and seven decisions $d_1, \cdots, d_7$. The decisions d_i ($i \le 5$) each have a 0.5 chance of causing outcome s_i and s_{i+1}, while d_6 causes either s_6 or s_1. Decision d_7, in contrast, is guaranteed to realize outcome s_7.

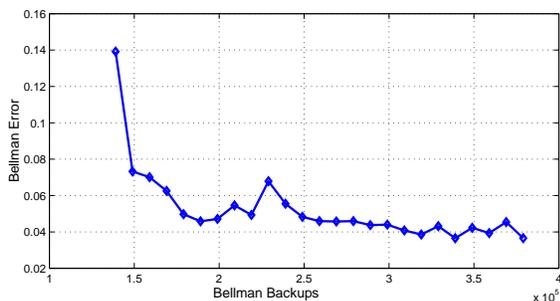Suppose our prior over utility functions is given by the mixture of uniforms with the fol-

---

[10]All experiments were implemented in Matlab, under Linux, using a PIII 933MHz, 512Mb PC.

Figure 4: Bellman Error: 20 outcomes, 30 decisions.

lowing six components (each weighted equally):

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
b_1	[.9 1]	[.9 1]	[0 .1]	[0 .1]	[0 .1]	[0 .1]	[.7 .8]
b_2	[0 .1]	[.9 1]	[.9 1]	[0 .1]	[0 .1]	[0 .1]	[.7 .8]
b_3	[0 .1]	[0 .1]	[.9 1]	[.9 1]	[0 .1]	[0 .1]	[.7 .8]
b_4	[0 .1]	[0 .1]	[0 .1]	[.9 1]	[.9 1]	[0 .1]	[.7 .8]
b_5	[0 .1]	[0 .1]	[0 .1]	[0 .1]	[.9 1]	[.9 1]	[.7 .8]
b_6	[.9 1]	[0 .1]	[0 .1]	[0 .1]	[0 .1]	[.9 1]	[.7 .8]

For each belief component b_i and state s_j, this table shows the range for which b_i assigns positive (uniform) density to $u(s_j)$. Intuitively, this prior reflects the fact that the user prefers some pair of (adjacent) outcomes from the set $\{s_1, \ldots, s_6\}$, but which exact pair is unknown. s_7 is considered to be a safe alternative. With this prior, myopic VOI associates no value to any query: a (noise-free) query can restrict the belief state to fewer than two components; but that will not be enough to change the optimal decision w.r.t. the prior (which is d_7). In contrast, the POMDP approach recognizes that the answers to a sequence of (properly chosen) queries can ensure that a better decision is made.

This problem was run using a similar training regime to those described above, with 98000 2-stage backups followed by 160000 full backups. Bellman error quickly falls to the 0.025 range (i.e., about 2.5%) after the two stage backups and hovers in that range for the remainder of the training run.[11] Backup up times for the 2-stage backups is 0.0014s per backup, while for full backups the time is 0.173s per backup. Two sample query paths for the specific prior above are shown in Figure 5. It should be noted that function approximation error can cause the value of queries to be overestimated on occasion. On a number of runs, the policy executed asks several questions about a specific utility dimension even though the current belief state contains enough information to recommend the optimal decision. We expect better function approximation strategies will help alleviate this problem. Again we point out that the myopic approach on this problem asks no questions for this prior and simply recommends the alternative, decision d_7.

As a final example, we consider a combinatorial bidding scenario, in which a bidding agent must offer bids for four different goods auctioned simultaneously in four different markets. To discretize the decision space, we assume that the

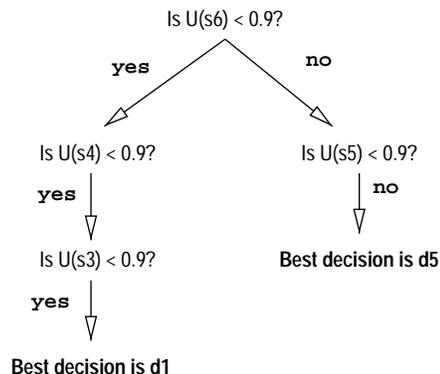[11]Query cost is 0.02 and noise probabilities are 0.03.

Figure 5: Example Query Paths: Myopic Problem.

agent can offer three distinct bids—low, medium, and high—for each good. Each of these bid levels corresponds to a precise cost: should the bid win, the user pays the price associated with each bid (with, of course, higher prices associated with higher bid levels). To suppress the need for strategic reasoning, the agent has a fixed, known probability of winning a good associated with each of the three bid levels. The probabilities of winning each good are independent, and increasing in the bid level.

With four goods and three bid levels, there are 81 possible decisions (mappings from markets to bids) and 16 outcomes (subsets of goods the user might obtain). The user's utility function need not be additive with respect to the goods obtained. For instance, the user might value goods g_1 and g_2 in conjunction, but may value neither individually. Thus utility is associated with each of the 16 outcomes. We assume that the overall utility function (accounting for the price paid for each good) is quasi-linear; so the price paid is subtracted from the utility of the subset of goods obtained.[12]

A plot of the Bellman error as a function of the backups for the bidding problem is shown in Figure 6 for a single run of the problem.

Our results are certainly preliminary, but do suggest that this approach to elicitation is feasible; since the model pushes almost all of the computational burden offline, computational concerns are mitigated to a large extent. However, our results also suggest the need for further study of suitable approximation architectures, and integration with parametric *utility models*. With parametric utility, large outcome spaces can be dealt with using low dimensional belief states, which will certainly enhance the feasibility of the model.

6 Concluding Remarks

We have described an abstract model of preference elicitation that allows for a system to optimally trade off the cost of elic-

[12]The specific parameter settings used are as follows. The prices associated with low, medium, and high bids are 0.02, 0.05 and 0.1, respectively (these are normalized on the utility scale $[0, 1]$). The probabilities of winning given a low, medium, and high bid, are 0.2, 0.4 and 0.85, respectively. These parameters are the same for all four markets. Query cost is 0.01 and responses have a 0.03 probability of being incorrect. The discount rate is 0.95.
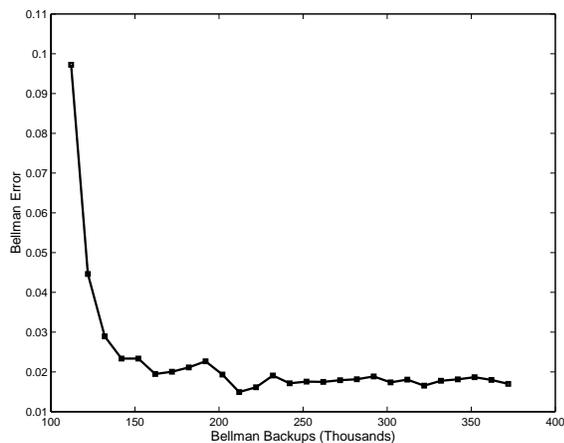
Figure 6: Bellman Error: Bidding Problem.

itation with the gain provided by elicited utility information. By casting the problem as a POMDP, the sequential and noisy nature of the process is addressed, as is the need to construct policies for arbitrary priors. Our function approximation techniques allow for optimization over continuous state and action spaces, and permit one to determine appropriate actions for mixture belief states despite the fact that we only compute value functions for single components.

We have described very preliminary experiments using specific utility, belief state, and approximation representations, but it should be clear that the general ideas apply much more broadly. Investigation of other instantiations of this model is ongoing. Of particular importance is the use of this approach with more compact utility functions representations (such as decomposed additive models), and exploiting independence in the belief state representation as well (which in turn enhances computation by reducing the inherent dimensionality of the Q-functions). Allowing more general queries is of obvious importance.

Maximizing "system utility" rather than "user utility" is an interesting variation of this model: in this case, knowledge of the user's utility function is useful if it enables good prediction of the user's behavior (e.g., purchasing patterns). The system then wishes to maximize its own utility (e.g., profit). One might also consider a game-theoretic extension in which a user wishes to conceal her preferences (to some extent) to foil this aim. Indeed, this form of elicitation could play an important role in automated negotiation and bargaining. Finally, effectively modeling long-term interactions (e.g., capturing utility functions that change over time), and learning suitable response and dynamics models, present interesting challenges.

References

[1] Dimitri P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, 1987.

[2] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon, Oxford, 1995.

[3] Craig Boutilier. On the foundations of *expected* expected utility. (manuscript), 2001.

[4] Craig Boutilier, Fahiem Bacchus, and Ronen I. Brafman. UCP-Networks: A directed graphical representation of conditional utilities. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 56–64, Seattle, 2001.

[5] U. Chajewska, L. Getoor, J. Norman, and Y. Shahar. Utility elicitation as a classification problem. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 79–88, Madison, WI, 1998.

[6] Urszula Chajewska, Daphne Koller, and Ronald Parr. Making rational decisions using adaptive utility elicitation. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 363–369, Austin, TX, 2000.

[7] Hsien-Te Cheng. *Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, University of British Columbia, Vancouver, 1988.

[8] Simon French. *Decision Theory*. Halsted Press, New York, 1986.

[9] Eric A. Hansen. Solving POMDPs by searching in policy space. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 211–219, Madison, WI, 1998.

[10] Milos Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, pages 33–94, 2000.

[11] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, New York, 1976.

[12] Long-Ji Lin and Tom. M. Mitchell. Memory approaches to reinforcement learning in non-Markovian domains. Technical Report CS–92–138, Carnegie Mellon University, Department of Computer Science, May 1992.

[13] Ronald Parr and Stuart Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1088–1094, Montreal, 1995.

[14] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[15] Sebastian Thrun. Monte Carlo POMDPs. In *Proceedings of Conference on Neural Information Processing Systems*, pages 1064–1070, Denver, 1999.