Scaling POMDPs for spoken dialog management

***Abstract*— Control in spoken dialog systems is challenging largely because automatic speech recognition is unreliable and hence the state of the conversation can never be known with certainty. Partially observable Markov decision processes (POMDPs) provide a principled mathematical framework for planning and control in this context; however, POMDPs face severe scalability challenges and past work has been limited to trivially small dialog tasks. This paper presents a novel POMDP optimization technique – composite summary point-based value iteration (CSPBVI) – which enables optimization to be performed on slot-filling POMDP-based dialog managers of a realistic size. Using dialog models trained on data from a tourist information domain, simulation results show that CSPBVI scales effectively, outperforms non-POMDP baselines, and is robust to estimation errors.**

***Index Terms*— Decision theory, dialogue management, partially observable Markov decision process, planning under uncertainty, spoken dialogue system.**

I. INTRODUCTION

SPOKEN dialog systems (SDSs) help people accomplish a task using spoken language. For example, a person might use an SDS to buy a train ticket over the phone, to direct a robot to clean a bedroom, or to control a music player in an automobile. Broadly, a spoken dialog system has three modules, one each for input, output, and control, shown in Figure 1. The input module performs automatic speech recognition (ASR) and spoken language understanding (SLU) to convert an acoustic speech signal into a hypothesis of the user's intention, such as *request(flight, from(london))*. The control module maintains an internal state and decides what action the machine should take. This includes communicative actions such as *ask(departure-date)* and non-communicative actions like consulting a database. The output module renders communicative actions from the machine as audio to the user.

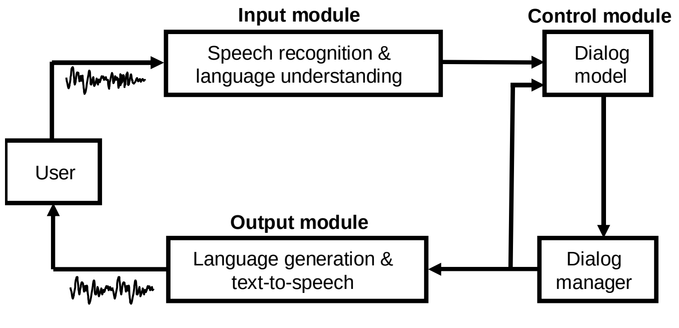
Input module
Control module
Speech recognition & language understanding
Dialog model
User
Output module
Language generation & text-to-speech
Dialog manager

Fig. 1. High-level architecture of a spoken dialog system.

This paper is concerned with the design of the control module. This is a challenging engineering problem in large part because automatic speech recognition (ASR) and understanding technology in the input module are error-prone, so a computer must regard everything it hears with suspicion. Worse, conflicting evidence does not always indicate a speech recognition error, because sometimes people change their objective in the middle of a conversation. Finally, conversation is a temporal process in which users behave in non-deterministic ways, and in which actions have both immediate and long-term effects. For all of these reasons, control in a spoken dialog system can be viewed as planning under uncertainty.

Past work has argued that partially observable Markov decision processes (POMDPs) provide a principled framework for control in spoken dialog systems [1], [2], [3], [4], [5], [6]. A POMDP views the state of the dialog as a hidden variable: rather than maintaining one hypothesis for the state of the dialog, a POMDP maintains a distribution over *all possible* dialog states, called a *belief state*. As the control module takes actions and receives evidence from the input module, this belief state is updated. Short- and long-term objectives of the system are specified in the form of the POMDP's reward function, and actions are selected with the goal of maximizing the sum of rewards *over time*: i.e., the POMDP performs planning to determine an optimal course of action which balances short-term and long-term priorities. Maintaining multiple hypotheses for the current dialog state enables POMDPs to better interpret conflicting evidence, and in the literature POMDPs have been shown to outperform (automated and hand-crafted) techniques which maintain a single dialog state hypothesis [1], [2], [3], [4], [5], [6].

Even so, POMDPs face severe scalability limitations. The computational complexity of planning grows astronomically as concepts are added to the dialog model, and POMDP-based spoken dialog systems in the literature have been limited to trivially small dialog domains. Worse, for reasons explained below, the existing techniques for scaling found in the POMDP literature are of little use in the dialog domain.

This paper presents a novel optimization technique for scaling POMDPs in the dialog domain called composite summary point-based value iteration (CSPBVI). Consideration is limited to so-called *slot-filling* dialogs, in which the machine seeks to collect a *value* for each of a set of *attributes* (or slots) as quickly and accurately as possible. To keep planning tractable, CSPBVI makes two important assumptions. First, machine actions are constrained to act on the single *best* hypothesis for each slot, regardless of its value. Planning then need only consider the proportion of probability mass held by this best hypothesis, reducing the size of the planning problem within each slot to a small constant. Second, an independence assumption is made across slots, allowing planning to be performed locally within each slot. At runtime each slot nominates an action, and a simple hand-crafted heuristic chooses which among these to take. With these assumptions, the complexity of the planning problem remains constant with respect to both the *number of values* per slot and the *number of slots*, allowing both to be increased to realistic, real-world sizes. Importantly, these assumptions affect *planning* but not *belief monitoring*: that is, they may impact plan quality, but they do not reduce the machine's ability to track the current state of the dialog or interpret conflicting evidence.

This paper is organized as follows. Section II reviews the fundamentals of POMDPs and reviews existing techniques for POMDP optimization. Section III reviews how POMDPs can be applied to the spoken dialog problem and explains why traditional optimization techniques fail in this domain. Section IV describes the CSPBVI method, and section V describes an example dialog management problem, demonstrates its operation, and compares its performance to a variety of baselines. Finally, section VI briefly concludes.

II. Background

This section reviews the definition of a partially observable Markov decision process (POMDP), explains exact optimization, and sketches approximate optimization techniques. For more detail and examples, refer to works such as [7], [8], [9], [10], [11], [12].

A. POMDP definition

Formally, a POMDP $\mathfrak{P}$ is defined as a tuple $\mathfrak{P} = (\mathbb{S}, \mathbb{A}, \mathbb{T}, \mathbb{R}, \mathbb{O}, \mathbb{Z}, \gamma, b_0)$ where $\mathbb{S}$ is a set of states s describing the machine's world with $s \in \mathbb{S}$; $\mathbb{A}$ is a set of actions a that a machine may take $a \in \mathbb{A}$; $\mathbb{T}$ defines a transition probability $P(s'|s,a)$; $\mathbb{R}$ defines the expected (immediate, real-valued) reward $r(s,a) \in \Re$; $\mathbb{O}$ is a set of observations o the machine can receive about the world with $o \in \mathbb{O}$; $\mathbb{Z}$ defines an observation probability $P(o'|s',a)$; γ is a geometric discount factor $0 < \gamma < 1$; and b_0 is an initial belief state, defined below.

The POMDP operates as follows. At each time-step, the world is in some unobserved state s. Since s is not known exactly, a distribution over states is maintained called a *belief state*, b. b is defined in *belief space* $\mathbb{B}$, which is an $(|\mathbb{S}|-1)$-dimensional simplex. $b(s)$ indicates the probability of being in a particular state s with initial belief state b_0. Based on b, the machine selects an action a, receives a reward $r(s,a)$, and transitions to (unobserved) state s', where s' depends only on s and a, according to $P(s'|s,a)$. The machine then receives an observation o' which depends on s' and a, according to $P(o'|s',a)$.

At each time-step, the belief state distribution b is updated using a function called the *state estimator* (*SE*), which computes a new distribution over states b' given a current distribution over states b, an action taken a, and an observation received o' [12]:

$$\begin{aligned} b'(s') &= SE(b,a,o') &(1)\\ &= \eta \cdot P(o'|s',a) \sum_s P(s'|a,s) b(s), &(2)\end{aligned}$$

where η is a normalization constant independent of b, a, and o' which ensures that $\sum_{s'} b'(s') = 1$.

As mentioned above, at each time-step, the machine receives reward $r(s,a)$. The cumulative, discounted reward is written as

$$V_t = \sum_{\tau=1}^{t} \gamma^{(\tau-1)} \sum_s b_\tau(s) r(s, a_\tau) \quad (3)$$

where $b_\tau(s)$ indicates the probability of being in state s at time-step τ and a_τ is the action a taken at time τ. The cumulative, discounted, *infinite horizon* reward is called the *return*, denoted by V_∞, or simply V for short. The goal of the machine is to choose actions in such a way as to maximize the return V given the POMDP parameters $\mathfrak{P} = (\mathbb{S}, \mathbb{A}, \mathbb{T}, \mathbb{R}, \mathbb{O}, \mathbb{Z}, \gamma, b_0)$. Determining how to choose actions in this context is called "optimization."

B. Exact POMDP optimization

Maximizing V in practice means finding a plan called a *policy* which indicates which actions to take at each turn. Policies can be formed as a collection of conditional plans. A *t-step conditional plan* describes how actions should be chosen for t steps into the future. Formally, a t-step conditional plan is a tree of uniform depth t and constant branching factor $|\mathbb{O}|$, in which each node is labelled with an action. The root node is referred to as layer t and the leaf nodes are referred to as layer 1. Every non-leaf node has $|\mathbb{O}|$ children, indexed as $1, 2, \ldots, |\mathbb{O}|$. A conditional plan is used to choose actions by first taking the action specified by the root node (layer t). An observation o will then be received from the POMDP, and control passes along arc o to a node in layer $t-1$. The action specified by that node is taken, and so on.

As an illustration, consider a POMDP with 2 observations and 4 actions. Three example conditional plans for this POMDP are shown in Figure 2. For example, conditional plan I specifies that action a_2 should be taken first. If o_1 is then received, a_3 should next be taken, and so on.

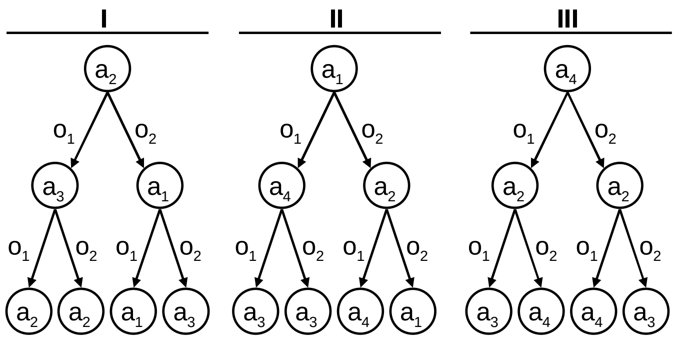
I
II
III
a2 o1 o2 a3 a1 o1 o2 o1 o2 a2 a2 a1 a3
a1 o1 o2 a4 a2 o1 o2 o1 o2 a3 a3 a4 a1
a4 o1 o2 a2 a2 o1 o2 o1 o2 a3 a4 a4 a3

Fig. 2. Three example 3-step conditional plans.

A t-step conditional plan has a *value* $V(s)$ associated with it, which indicates the expected value of the conditional plan, depending on the current (unobserved) state s. This value can be calculated recursively for $\tau = 1 \ldots t$ as:

$$\begin{aligned} V_\tau(s) &= r(s,a_\tau) + \gamma \sum_{s'} P(s'|s,a_\tau) \cdot \\ &\quad \sum_{o'} P(o'|s',a_\tau) V_{\tau-1}^{o'}(s'), &(4)\end{aligned}$$

where a_τ gives the action associated with the root node of this conditional plan, $V_{\tau-1}^{o'}$ indicates the value of the conditional plan in layer $\tau - 1$ which is the child index o' of the conditional plan, and $\forall s, V_0(s) = 0$. Continuing the example above, suppose the POMDP has 2 states and the values of each of the conditional plans were found to be:

$$\begin{aligned} V_I &= (4.0, 3.0) \\ V_{II} &= (3.3, 3.3) \\ V_{III} &= (3.0, 4.0) \end{aligned} \tag{5}$$

Then if plan III were executed repeatedly from state s_2, the average discounted sum of the 3 rewards obtained would be 4.0.

Of course at runtime, the machine does not know the state s exactly and instead maintains a belief state b. The value of a conditional plan at a belief state b is then computed as an expectation over states:

$$V(b) = \sum_s b(s)V(s). \tag{6}$$

In a POMDP, the machine's task is to choose between a number of conditional plans to find the one which maximizes V_t for some b. Given a set of t-step conditional plans $\mathcal{N}_t$ with $n \in \mathcal{N}_t$, and their corresponding values $\{V_t^n\}$ and initial actions $\{a_t^n\}$, the value of the best plan at belief state b is:

$$V^*_{\mathcal{N}_t}(b) = \max_n \sum_s b(s)V_t^n(s). \tag{7}$$

$V^*_{\mathcal{N}_t}(b)$ implies an *optimal policy* $\pi^*_{\mathcal{N}_t}(b)$:

$$\pi^*_{\mathcal{N}_t}(b) = a_t^n \text{ where } n = \arg\max_n \sum_s b(s)V_t^n(s). \tag{8}$$

In words, $V^*_{\mathcal{N}_t}(b)$ represents the (scalar) expected value of starting in b and following the best t-step conditional plan in $\mathcal{N}_t$, which begins with action $\pi^*_{\mathcal{N}_t}(b)$.

This process is illustrated graphically in Figures 3 and 4. In this example, $\mathcal{N}_3$ consists of the three 3-step conditional plans I, II, and III given in equation (5). In this depiction, the horizontal axis is the belief state b, where the left end of the axis indicates that $b = (1, 0)$ (i.e., state s_1 with certainty) and the right end indicates $b = (0, 1)$ (i.e., state s_2 with certainty). The vertical axis shows the values V of the conditional plans. In Figure 3, the values of the three conditional plans are shown with the dotted lines. In this 2-state example, the value functions can be shown as lines, but in general the values of conditional plans are hyperplanes in $(|\mathbb{S}| - 1)$-dimensional space.

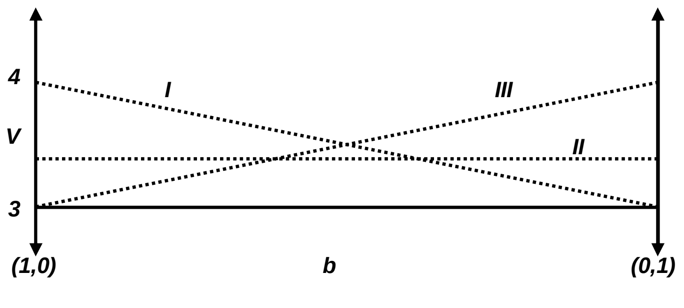

Fig. 3. Value functions for the three conditional plans shown in Figure 2.

Figure 4 shows the *optimal* value function, which is the upper surface of all of the value functions, shown as the heavy line. Note how there are two regions: on the left where conditional plan I is optimal and on the right where conditional plan III is optimal.

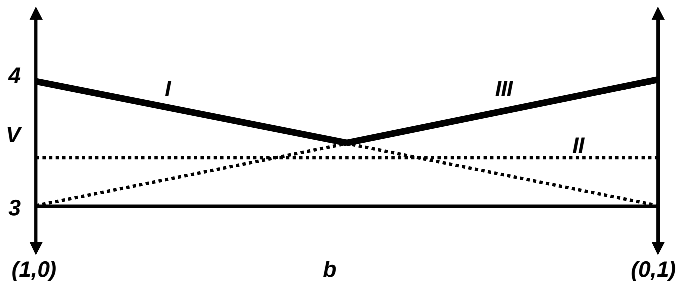

Fig. 4. Optimal value function (shown as the heavy line) for the three conditional plans I, II, and III shown in Figure 2.

If $\mathcal{N}_t$ contained all possible conditional plans, then $V^*_{\mathcal{N}_t}(b)$ would give the value of the *optimal t-step policy* for this POMDP. Unfortunately, this type of exhaustive approach is doomed to failure since the number of possible t-step conditional plans is

$$|A|^{\frac{|O|^t - 1}{|O| - 1}} \tag{9}$$

which grows astronomically in t. In the example here with 2 states, 4 actions, and 2 observations, there are already 16,384 distinct 3-step conditional plans.

Fortunately, it has been found empirically that relatively few t-step conditional plans make a contribution to an optimal t-step policy. For example, in Figure 4, there are no belief points for which conditional plan II is optimal (i.e., forms the upper surface). As a result, conditional plan II will never contribute to an optimal 3-step policy and can safely be discarded. In general, this insight can be exploited to compute optimal policies more efficiently with *value iteration* [11], [9]. Value iteration is an exact, iterative, dynamic programming process in which successively longer planning horizons are considered, and an optimal policy is incrementally created for longer and longer horizons. Value iteration proceeds by finding the *subset* of possible t-step conditional plans which contribute to the optimal t-step policy. These conditional plans are called *useful*, and only useful t-step plans are considered when finding the $(t + 1)$-step optimal policy. For example, in Figure 4, conditional plans I and III are useful; conditional plan II is not and will never form a child of a 4-step conditional plan.

Each iteration of value iteration consists of two steps. First, in the "generation" step, all possible t-step conditional plans are created by enumerating all actions followed by all possible useful combinations of $(t - 1)$-step plans, producing $|A||\mathcal{N}_{t-1}|^{|O|}$ t-step plans. Then, in the "pruning" step, conditional plans which do not contribute to the optimal t-step policy are removed, leaving the set of useful t-step plans. The algorithm is repeated for T steps, and produces the values and initial actions of the optimal T-step policy. For sufficiently large T it can be shown that a policy produced in this way converges to the optimal infinite horizon policy [12].

The pruning operation seeks to eliminate conditional plans which do not contribute to the optimal policy. One way this can be implemented is with linear programming [12], which seeks to find a belief point for which a conditional plan is optimal, or to determine that such a point does not exist. The complexity of this linear program is significant and grows with the dimensionality of belief space $|\mathbb{S}|$. In practice, the combination of growth in the number of conditional plans, and the computational complexity of the "pruning" operation cause exact value iteration to be intractable for problems on the order of 10 states, actions, and observations.[1] Therefore to scale POMDP policy optimisation to real-world tasks, *approximations* must be made to the exact optimal policy. This is described next.

C. Approximate POMDP optimization

Exact value iteration is computationally complex primarily because it attempts to find an optimal policy for *all points* in belief space $\mathcal{B}$. A family of approximate techniques quantizes belief space into a set of points and only attempts to find optimal plans at these points.

Point-based value iteration (PBVI) is an example of one such technique [13], [14].[2] PBVI first generates a set of N belief points PBVI $\hat{\mathcal{B}} = \{b_1, b_2, \ldots, b_N\}$ by following a random policy to sample belief space. In theory it is possible that any belief point might eventually be reached starting from b_0, but in practice this is rare and the belief point selection process attempts to find those belief points which are likely to be reached. Then, value iteration is performed, but rather than searching all of belief space for vectors to prune, PBVI takes a simple max operation at each belief point. Like exact value iteration, PBVI produces a set of vectors $\mathcal{N}$ and corresponding actions. However, unlike exact value iteration the number of vectors produced in each iteration is constant, because each vector corresponds to a belief point b_n in $\hat{\mathcal{B}}$. Furthermore the conditional plan found for each belief point b_n is only guaranteed to be optimal for that belief point. However, the hope is that it will be optimal, or nearly so, at other points nearby. At runtime, an optimal action a may be chosen for any belief point b by evaluating $a = a_n$ where $n = \arg\max_n \sum_s b(s) V^n(s)$, just as in exact value iteration.

The value of each of these conditional plans, $V_t^n(s)$, is *exact*, but only guaranteed to be *optimal* at b_n, and in this respect PBVI is an approximation technique. As more belief points are added, the quality of optimization increases at the expense of additional computational cost, allowing trade-offs to be made between optimization quality and computational cost [13], [14]. Moreover, since a vector (or, stated equivalently, a value *and* a value gradient) is maintained for each belief point, interpolation for unsampled belief points at runtime can be done accurately. PBVI has been demonstrated to find good policies on POMDPs with thousands of states.

In summary, conditional plans provide a framework for evaluating different courses of action, but enumerating all possible conditional plans is hopelessly intractable. Value iteration builds conditional plans incrementally for longer and longer time horizons, discarding useless plans as it progresses, making policy optimization possible for small POMDPs. Even so, optimization with exact value iteration quickly becomes infeasible and approximate techniques such as point-based value iteration (PBVI) provide a way to trade off computational complexity and plan quality, scaling to larger problems.

III. POMDPs AND DIALOG MANAGEMENT

This section first explains how spoken dialog systems may be cast as a POMDP, and then shows why even approximate optimisation methods rapidly become intractable when scaled to real problems in this domain.

The techniques in this paper are based on the SDS-POMDP model which has been previously presented in [4], [5], and [6], and which is reviewed here for completeness. In the SDS-POMDP, the POMDP state variable s is separated into three components, $s = (s_u, a_u, s_d)$. The component $s_u \in \mathbb{S}_u$ gives the *user's goal*, such as a complete travel itinerary. This paper is concerned with so-called slot-filling dialogs in which the user's goal s_u is composed of W *slots*, $s_u = (s_u^1, \ldots, s_u^W)$, where $s_u^w \in \mathcal{S}_u^w$. For example, in the air travel domain, a user goal s_u might be composed of $s_u = (s_u^{\text{from}}, s_u^{\text{to}}, s_u^{\text{class}}, s_u^{\text{airline}}, s_u^{\text{time}}, s_u^{\text{date}})$. The component $a_u \in \mathbb{A}_u$ gives the most recent *user action* at the concept level, such as stating a place the user would like to travel to, responding to a yes/no question, or a "null" response indicating the user was silent. Finally the component $s_d \in \mathbb{S}_d$ records relevant *dialogue history*, such as the grounding status of a slot, or how many times a slot has been queried. None of these components is observable directly by the machine and the SDS-POMDP belief state is formed from a distribution over these components $b(s_u, a_u, s_d)$.

The POMDP action a corresponds to the machine action in the dialog, such as greeting the user, asking the user where they want to go "to", or confirming a user goal. Finally, the POMDP observation o is separated into two components $o = (\tilde{a}_u, c)$, where $\tilde{a}_u \in \mathbb{A}_u$ gives the hypothesis of the user's action provided by the speech recognition process, and c is a confidence score.

By substitution and making reasonable conditional independence assumptions, the POMDP transition function $P(s'|s, a)$ and observation function $P(o'|s', a)$ can be re-written in SDS-POMDP terms as $P(s'|s, a) = P(s'_u|s_u, a)P(a'_u|s'_u, a)P(s'_d|s'_u, a'_u, s_d, a)$ and $P(o'|s', a) = P(\tilde{a}'_u, c'|a'_u)$. These individual probability functions correspond to intuitive *models* which can either be estimated from data or handcrafted. For example, $P(a'_u|s'_u, a)$ provides a model of user behavior which can be estimated from dialog data, and $P(s'_d|s'_u, a'_u, s_d, a)$ could be handcrafted following e.g., the Information State Update approach [15]. The design of the reward function $r(s_u, a_u, s_d, a)$ is left to the application

[1]Technically it is the *complexity* of optimal policies, and not the number of states, actions, and observations which causes value iteration to become intractable, but it is not obvious how to calculate the complexity of a plan *a priori* and in practice the number of states, actions, and observations is a useful heuristic.

[2]The phrase "Point-based value iteration" and acronym PBVI were coined by Pineau [13]. Subsequent work extended Pineau's formulation [14], and in this paper PBVI refers to this family of techniques.

designer as it implements the design objectives of a given system. In general r encodes trade-offs between speed, appropriateness, and accuracy, and one would expect these to be different in (for example) the banking and entertainment domains.

The slot-filling SDS-POMDP model may be regarded as an extension of a "frame-based" dialog system [16], in which dialog state is a frame of attributes that are populated with values as the dialog progresses.[3] Actions are selected by a (hand-crafted) dialog management algorithm that examines the frame and determines which element to query or confirm next. Users can provide values for any slot in an utterance, such as "Where are you flying to?" / "To New York on Monday", and in this respect frame-based approaches support a limited form of mixed initiative.

A slot-filling SDS-POMDP model extends the frame-based approach by maintaining not a single frame, but rather a distribution (belief state) over *all possible* frames. Instead of populating values as evidence arrives, the belief state sharpens around the frame that is most likely. Further, action selection is performed using an optimization criterion rather than a hand-crafted dialog manager. The optimization process produces a dialog manager (policy) that selects actions to maximize the sum of rewards gained over the course of the dialog. In other words, the optimization process performs planning to find the best action to take for a given distribution over frames (belief state).

Despite the theoretical appeal of the SDS-POMDP in practice optimization faces severe scalability issues. For example, if the size of each slot is $|\mathcal{S}_u^w| = 100$, then there are a total of $|\mathbb{S}_u| = \prod_w |\mathcal{S}_u^w| = 100^W$ distinct user goals. Because the set of user actions $\mathbb{A}_u$ and machine actions $\mathbb{A}$ often refer to specific user goals, (for example, a user action which states part of a goal such as "A flight to Paris, please", or a machine action which confirms part of a goal such as "Leaving from London, is that right?"), the SDS-POMDP action and observation sets all grow with the number of user goals. A 5-slot problem where each slot has 100 slot values is considered small in the dialog community, yet it is completely intractable for state-of-the-art approximate POMDP optimization techniques such as *Perseus* [14]. In fact, letting G equal the number of values for each slot, the number of plans required by PBVI algorithms like *Perseus* grows as $O(G^W)$. While this represents a significant improvement over value iteration which requires at worst $O[(G^W)^{(G^W)^T}]$ conditional plans, it still grows exponentially in the number of slots.[4]

A common scaling technique in the POMDP literature is to "compress" the state space by aggregating states [20], [21], [22], [23]. Unfortunately, in the dialog domain, there is an important correspondence between states and actions, and this correspondence would be lost in a compression. For example a user goal such as $s_u =$ *from-london* has corresponding system actions such as *confirm-from-london* and *print-ticket-from-london*, and it seems unlikely that an aggregated state such as *from-london-or-leeds* would be helpful. As a result, optimization techniques which attempt to compress the POMDP through state aggregation are bound to fail in the dialog domain. Similarly, attempting to exploit the factored form of the SDS-POMDP in optimization (using e.g., [24], [25]) is unlikely to succeed since most of the growth is due to one component (the user's goal), and the number of required conditional plans required grows with each user goal. As a result, to maintain performance, the complexity of an approximation will grow as user goals are added.

Therefore to realize the benefits of the POMDP model in the dialog domain, a new optimization method is needed.

IV. CSPBVI METHOD DESCRIPTION

A. Intuition

Composite summary point-based value iteration (CSPBVI) is a novel POMDP optimization technique which enables a slot-filling SDS-POMDP to be scaled to a realistic size.[5] To do this, CSPBVI makes two important assumptions.

First, looking through transcripts of simulated dialogs with much smaller POMDP-based dialog applications, it was noticed that actions which operate on a user goal like *confirm* or *print-ticket* were only taken on the user goal with the highest belief mass. Intuitively, this is sensible: for confirmations, the machine should minimize the chances of a "no" response as this increases belief state entropy, lengthens the dialog, and decreases return. Moreover, committing to the wrong user goal when closing the dialog (e.g., printing a ticket) results in severe penalties. With this in mind, the first assumption made by CSPBVI is to limit *a priori* actions like *confirm* and *print-ticket* to act on only the most likely user goal. Then, planning considers only the *proportion* of belief mass held by the most likely user goal (and not its actual value). The structure of the slot-filling domain provides the framework required to map between actions and user goals.

Second, in a recent data collection [28], it was noticed that when users are asked about a certain slot, they most often provide a value for just that slot, and only sometimes provide values for other slots.[6] CSPBVI capitalizes on this insight by assuming that cross-slot effects are unimportant for planning. Hence, it first estimates system dynamics locally for each slot, then uses these estimates to produce a distinct dialog manager (i.e., POMDP policy) for each slot. In effect, actions are chosen based on the expectation that user responses will not provide information about other slots. At runtime, each dialog manager nominates an action appropriate for its slot and a handcrafted heuristic chooses which one of these to take.

[3]In principle the general SDS-POMDP model is comparable to the more advanced "agent-based" model in which the dialog manager maintains beliefs about the dialog and the user's goals, and performs planning to achieve its goals [16]. However the practicalities of this level of generality are not addressed in this paper.

[4]An alternative is to abandon planning altogether and greedily select actions [17], [18], [19]. While this avoids the computational problems of planning, it requires that the designer somehow encode incentives into the reward function to make long-term progress toward a goal, which inevitably requires some hand-crafting and iterative tuning.

[5]CSPBVI and a precursor were previously described in workshop papers by the authors [26], [27].

[6]See table II for data.

Together, these two assumptions reduce complexity significantly: plan growth is constant with respect to the number of slot values G, and is performed separately for each slot, resulting in W separate optimizations, each with a $O(c)$ plans, where c is a constant with respect to the number of slots (W), the number of values per slot (G), and the planning horizon (T). Moreover, while these two assumptions affect planning, they do not alter how evidence is interpreted: CSPBVI still performs belief monitoring over all user goals. Hence, when a user *does* provide extra information or when conflicting evidence is received, it is still properly incorporated into the belief state.

B. Description

CSPBVI consists of four phases: construction, sampling, optimization, and execution. In the *construction* phase, first the *master* POMDP is created, which is an SDS-POMDP with several constraints and additions. The user's goal $s_u \in \mathbb{S}_u$ is decomposed into W slots, $s_u = (s_u^1, \ldots, s_u^W)$ where $s_u^w \in \mathcal{S}_u^w$ and where $\mathcal{S}_u^w$ refers to the set of values for slot w. The dialog history $s_d \in \mathbb{S}_d$ is similarly decomposed into W slots, $s_d = (s_d^1, \ldots, s_d^W)$ where $s_d^w \in \mathcal{S}_d^w$ and where $\mathcal{S}_d^w$ refers to the set of possible dialog histories for slot w. Machine actions are formed of predicates which take arguments that encode the slot w and the value (or values) s_u^w to which the action refers. Machine actions are written *predicate*$[w](x)$, where *predicate* refers to the illocutionary force of the action, w refers to a slot index, and x refers to the slot value(s) referred to by the action, if any. For example, the SDS-POMDP machine actions *ask-from* and *confirm-to-london* would be restated as *ask*[*from*]() and *confirm*[*to*](*london*). A special meta-slot $w = \textit{all}$ denotes an action which refers to all slots, such as $\textit{submit}[\textit{all}](s_u^{\text{from}} = \textit{london}, s_u^{\text{to}} = \textit{paris})$ and $\textit{greet}[\textit{all}]()$. Finally, in the master POMDP a modified reward function is created $r_w(s,a)$ which removes conditioning on all but the slot w. For example, if the reward r for incorrectly/correctly submitting a user's *complete* goal is $-25/{+25}$, then r_w would be assigned $-25/{+25}$ for incorrectly/correctly submitting *only* slot w, ignoring all others. Also, belief monitoring must be tractable in the master POMDP, and this may require approximations in the observation function; an example of this is shown in the next section.

After the master POMDP is formed, W belief Markov decision processes (BMDPs) are constructed.[7] Each of these will provide a compact representation of the belief state of a single slot, and for this reason they will be called "Summary BMDPs". Each of these has a state space with two components, $\hat{\mathcal{S}}_u^w$ and $\hat{\mathcal{S}}_d^w$, where $\hat{\mathcal{S}}_u^w = \{\textit{best}, \textit{rest}\}$ and $\hat{\mathcal{S}}_d^w = \mathcal{S}_d^w$. A belief point in master space b can be mapped to a belief point in summary space $\hat{b}$ for a specific slot w with the function *bToSummary* (Algorithm 1). This function sets the summary belief component $\hat{b}(\hat{s}_u^w = \textit{best})$ equal to the probability mass of the most likely user goal in slot w, sets $\hat{b}(\hat{s}_u^w = \textit{rest})$ equal to the remaining probability mass of all the rest of the user goals in slot w, and sets the dialog history in summary space $\hat{b}(\hat{s}_d^w)$ equal to the dialog history in master space $b(\hat{s}_d^w)$.

Algorithm 1: Function *bToSummary*.

```
Input: $b$, $w$
Output: $\hat{b}$ (for slot $w$)
1 $\hat{b}(\hat{s}_u = \textit{best}) \leftarrow \max_{s_u^w} b(s_u^w)$
2 $\hat{b}(\hat{s}_u = \textit{rest}) \leftarrow 1 - \hat{b}(\hat{s}_u^w = \textit{best})$
3 foreach $s_d^w \in \mathcal{S}_d^w$ do
4     $\hat{b}(s_d^w) \leftarrow b(s_d^w)$
```

Algorithm 2: Function *aToMaster*.

```
Input: $\hat{a}$, $\hat{w}$, $w$, $b$
Output: $a$
1 if $\hat{a}$ operates on all slots then
2     for $\tilde{w} \leftarrow 1$ to $W$ do
3         $s_u^{\tilde{w}} \leftarrow \arg\max_{\tilde{s}_u^{\tilde{w}}} b(\tilde{s}_u^{\tilde{w}})$
4     $a \leftarrow \hat{a}[\text{all}](s_u^1, \ldots, s_u^W)$
5 else
6     if $\hat{w} = \textit{this}$ then
7         $w^* = w$
8     else
9         $w^* = \text{randIntOmit}(W, w)$
10    if $\hat{a}[w^*]$ takes an argument in master space then
11        $s_u^{w^*} \leftarrow \arg\max_{s_u^{w^*}} b(s_u^{w^*})$
12        $a \leftarrow \hat{a}[w^*](s_u^{w^*})$
13    else
14        $a \leftarrow \hat{a}[w^*]()$
```

The action set of each of these summary BMDPs consists of the predicates of $\mathbb{A}$ and take one argument, $\hat{w} \in \{\textit{this}, \textit{other}\}$, where $\hat{w} = \textit{this}$ indicates that an action in master space a refers to *this* slot and $\hat{w} = \textit{other}$ indicates that a refers to *some other* slot. (If the action a operates on *all* slots, $\hat{w}$ is set to *this*.) For example, in a slot-filling SDS-POMDP with two slots *from* and *to*, a master POMDP action $a = \textit{confirm}[\textit{from}](\textit{london})$ would be mapped to $\hat{a}^{\text{from}} = \textit{confirm}[\textit{this}]$ in the summary BMDP for the *from* slot, and $\hat{a}^{\text{to}} = \textit{confirm}[\textit{other}]$ in the summary BMDP for the *to* slot.

Actions in summary space are mapped into master space by appending the most likely user goal, and this mapping is implemented by the function *aToMaster* (Algorithm 2). The $\arg\max$ operation in line 11 of Algorithm 2 implements the central CSPBVI assumption that actions like "confirm" or "print-ticket" are limited to the most likely user goal.

For reference, the components of the master POMDP and summary BMDPs for a two-slot dialog task in the travel domain are shown in table I.

The *sampling* phase of CSPBVI is shown in Algorithm 5.[8] Sampling iterates over each slot $w = 1 \ldots W$, and for each slot consists of 3 stages. In the first stage (Algorithm 5, lines 2-19),

[7]A *Belief MDP* is a Markov decision process with a continuous state corresponding to a POMDP belief state [12].

[8]Algorithms 5-7, which involve the sampling process, are listed in the appendix.

TABLE I
COMPONENTS OF THE MASTER POMDP AND SUMMARY BMDPS FOR A TWO-SLOT DIALOG TASK IN THE TRAVEL DOMAIN.

	Master POMDP	Summary BMDP for *from* slot	Summary BMDP for *to* slot
State	$(s_u^{\text{from}}, s_u^{\text{to}}, a_u, s_d^{\text{from}}, s_d^{\text{to}})$	$(\hat{s}_u^{\text{from}}, \hat{s}_d^{\text{from}})$	$(\hat{s}_u^{\text{to}}, \hat{s}_d^{\text{to}})$
State space size	$\approx \lvert\mathcal{S}_u^{\text{from}}\rvert \cdot \lvert\mathcal{S}_u^{\text{to}}\rvert \cdot \lvert\mathcal{S}_d^{\text{from}}\rvert \cdot \lvert\mathcal{S}_d^{\text{to}}\rvert = 1000^2 \cdot 3^2$	$= \lvert\hat{\mathcal{S}}_u^{\text{from}}\rvert \cdot \lvert\hat{\mathcal{S}}_d^{\text{from}}\rvert = 2 \cdot 3$	$= \lvert\hat{\mathcal{S}}_u^{\text{to}}\rvert \cdot \lvert\hat{\mathcal{S}}_d^{\text{to}}\rvert = 2 \cdot 3$
Action	a	$\hat{a}^{\text{from}}$	$\hat{a}^{\text{to}}$
Action space size*	$\approx 1000^2$	$= 3$	$= 3$
Belief state	$b(s_u^{\text{from}}, s_u^{\text{to}}, a_u, s_d^{\text{from}}, s_d^{\text{to}})$	$\hat{b}(\hat{s}_u^{\text{from}}, \hat{s}_d^{\text{from}})$	$\hat{b}(\hat{s}_u^{\text{to}}, \hat{s}_d^{\text{to}})$
Sampled quantities	—	$\{\hat{b}_{\text{from},n}\}, \{l(\text{from}, n, \hat{a}, k)\}, \{\hat{r}_{\text{from},n}^{\hat{a},k}\}$	$\{\hat{b}_{\text{to},n}\}, \{l(\text{to}, n, \hat{a}, k)\}, \{\hat{r}_{\text{to},n}^{\hat{a},k}\}$

*In the master POMDP, there is one *submit* action for each distinct user goal (i.e., *from-to* city pair); with 1000 cities there are approximately 1000^2 distinct actions. By contrast, in each summary BMDP, there are 3 actions: *ask, confirm,* and *submit.*

for each w the machine takes actions randomly to sample N points in summary space, written as the set $\{\hat{b}_{w,n}\}$. Initially this set is empty and at each step $n = 1 \ldots N$, the current belief point b is mapped into summary space for slot w to produce $\hat{b}$ with *bToSummary*. If $\hat{b}$ (or a point close to $\hat{b}$) has not already been visited, then it is added and two other quantities are sampled using *samplePoint* (Algorithm 6). *samplePoint* takes each summary action K times, $k = 1 \ldots K$, resetting to b after each, and recording the resulting reward in $\hat{r}_{w,n}^{\hat{a},k}$ and the successor point in summary space in $\hat{b}_{w,n}^{\hat{a},k}$. In the second stage of sampling (Algorithm 5, lines 20-28), sampling is repeated for the *corners* of summary space for each slot to help ensure coverage of summary space (Algorithm 7). In the third stage (Algorithm 5, lines 29-32), for each point $\hat{b}_{w,n}^{\hat{a},k}$, the closest point in $\hat{b}_{w,n}$ is located and its index is recorded in $l(w, n, \hat{a}, k)$. In summary, the sampling phase produces three quantities: $\{\hat{b}_{w,n}\}$ which is a set of sampled points in summary space for each slot w; $\{l(w, n, \hat{a}, k)\}$ which are the indices of the closest points in $\{\hat{b}_{w,n}\}$ when action $\hat{a}$ was taken from $\hat{b}_{w,n}$ the kth time; and $\{\hat{r}_{w,n}^{\hat{a},k}\}$ which is the reward obtained when action $\hat{a}$ was taken from $\hat{b}_{w,n}$ the kth time.

CSPBVI *optimization*, shown in Algorithm 3, is run W times, once for each slot w using that slot's dynamics and reward. Dynamic programming is used to iteratively find the best action and its expected value at each belief point $\hat{b}_{w,n}$ for longer and longer planning horizons. Each iteration first computes $\hat{q}^{\hat{a},n}$, which estimates the value of taking action $\hat{a}$ from point $\hat{b}_{w,n}$, then from this computes $\hat{a}_t^{w,n}$ (the optimal t-step action at $\hat{b}_{w,n}$) and $\hat{v}_t^{w,n}$ (the expected value of the optimal t-step policy starting from $\hat{b}_{w,n}$). Summary actions selected in each iteration are *restricted* to $\hat{w} =$ *this*: that is, only actions which operate on *this* slot (or all slots) are incorporated into conditional plans. Optimization ultimately produces an optimal summary action $\hat{a}^{w,n}$ for each point $\hat{b}_{w,n}$.

A CSPBVI policy is *executed* as shown in Algorithm 4. Belief monitoring is performed in the master POMDP, and for a given belief point b in master space, the corresponding set of summary belief points $\hat{b}_w$ is computed for all slots w. For each belief point $\hat{b}_w$ the index of the closest point n^* in the set $\{\hat{b}_{w,n}\}$ is found, and its summary action ($\hat{a}^{w,n^*}$) is mapped to a master action a^w. This process is repeated for each slot w and produces a vector of nominated master actions, a^w. Finally, a handcrafted heuristic called *chooseActionHeuristic*, which must be created for each application, selects an action from this vector to take.

Algorithm 3: CSPBVI optimization procedure.

```
   Input: $\mathfrak{P}_{SDS}$, $\{\hat{b}_{w,n}\}$, $\{l(w,n,\hat{a},k)\}$, $\{\hat{r}_{w,n}^{\hat{a},k}\}$, $K$, $T$
   Output: $\{\hat{a}_t^{w,n}\}$
1  for $w \leftarrow 1$ to $W$ do
2      $N \leftarrow |\{\hat{b}_{w,n}\}|$
3      for $n \leftarrow 1$ to $N$ do
4          $\hat{\text{v}}_0^n \leftarrow 0$
5      for $t \leftarrow 1$ to $T$ do
           // Generate $\{\hat{q}^{\hat{a},n}\}$, values of all
           // possibly useful CPs.
6          for $n \leftarrow 1$ to $N$ do
7              foreach $\hat{a} \in \hat{\mathcal{A}}^w$ do
8                  $\hat{w} \leftarrow$ this
9                  $\hat{q}^{\hat{a},n} \leftarrow \frac{1}{K}\sum_k \hat{r}_{w,n}^{\hat{a}[\hat{w}],k} + \gamma \hat{\text{v}}_{t-1}^{l(w,n,\hat{a}[\hat{w}],k)}$
           // Prune $\{\hat{q}^{\hat{a},n}\}$ to yield $\{\hat{\text{v}}_t^n\}$,
           // values of actually useful CPs.
10         for $n \leftarrow 1$ to $N$ do
11             $\hat{a}^* \leftarrow \arg\max_{\hat{a}} \hat{q}^{\hat{a},n}$
12             $\hat{a}_t^{w,n} \leftarrow \hat{a}^*$
13             $\hat{\text{v}}_t^n \leftarrow \hat{q}^{\hat{a}^*,n}$
```

Because the number of summary actions and summary states are constant with respect to the number of slots (and the number of values for each slot), CSPBVI optimization scales to handle many slots. The quality of the solution produced is a function of the optimization parameters T, N, and K, and of the quality of the handcrafted action selection heuristic.

However, the assumptions which allow CSPBVI to scale introduce three potential limitations. First, like PBVI, CSPBVI optimizes actions for a finite set of points $\{\hat{b}_n\}$ and not the entire belief simplex. As such it is always possible that a conditional plan which is optimal for a region which does not include a belief point will be omitted. In practice there are relatively few summary actions and thus (most likely) relatively few regions, so provided enough points are sampled it seems improbable that a region would fail to be included.

Second, since the summary belief state is a non-linear function of the master belief state, the dynamics of summary space are not guaranteed to be Markovian. As a result, the

Algorithm 4: CSPBVI action selection procedure, used at runtime.

```
Input: $b$, $\{\hat{b}_{w,n}\}$, $\{\hat{a}_{w,n}\}$
Output: $a$
1 for $w \leftarrow 1$ to $W$ do
2     $\hat{b} \leftarrow$ bToSummary$(b, w)$
3     $n^* \leftarrow \arg\min_n |\hat{b}_{w,n} - \hat{b}|$
4     $a^w \leftarrow$ aToMaster$(\hat{a}^{w,n^*}, \hat{w} = \textit{this}, b, w)$
5 $a \leftarrow$ chooseActionHeuristic$(\{a^w\})$
```

central Markov assumption of value iteration may be violated and value iteration may fail to produce good policies.[9] Yet in general, reinforcement learning has been found to usually produce good plans in non-Markovian environments [29].

Finally, CSPBVI relies on the assumption that a "myopic" view of system dynamics local to each slot is sufficient for planning, and that a handcrafted heuristic can successfully choose actions. This assumption seems well-founded since experience from past work in dialog management suggests that good plans can be constructed by operations which consider the state of a single slot at a time.

In sum, our intuition is that these theoretical limitations will not be a problem in practice, and the next section tests CSPBVI empirically to verify performance on a slot-filling dialog task of a realistic size.

V. Evaluation

In this section, CSPBVI is evaluated in several ways. Traditionally dialog managers have been designed by hand; more recently (fully-observable) Markov decision process (MDP) have been shown to provide gains over hand-crafted designs. In this section, CSPBVI is compared to an MDP-based baseline and a handcrafted baseline, and is shown to out-perform both. Further evaluations show that CSPBVI scales better than standard POMDP optimization with little loss in performance.

Because the aim of these evaluations is to show that statistically significant performance gains hold across a variety of operating conditions (such as the number of slots and the speech recognition error rate), hundreds of thousands of dialogs are needed, and conducting these dialogs with real users would be impossible. Rather, a simulated user has been created based on real dialog data. Past work has shown that performance improvements predicted by user models in reinforcement-learning based systems are borne out when deployed to real users [30]. Further, in this evaluation, policies trained on the simulated user are evaluated on a second simulated user estimated from held-out data, and it is found that performance does not degrade significantly. This provides some assurance that the learned policies are robust to variations in the parameters of the simulated user.

A. Example spoken dialog system

A POMDP-based dialog manager called MaxiTravel-W was created. MaxiTravel-W is an SDS-POMDP with W slots, where each slot contains 100 values and where W can be varied. The user's (single intention) goal consists of a single value for each slot. To keep belief monitoring tractable, some independence assumptions between slots are made. User actions are decomposed by slot into $a_u = (a_u^1, \ldots, a_u^W)$, and each per-slot user action element a_u^w is decomposed into three components $a_u^w = (a_{\text{state}}^w, a_{\text{stateSlot}}^w, a_{\text{yesNo}}^w)$, where $a_{\text{state}}^w \in \mathcal{A}_{\text{state}}^w$, $a_{\text{stateSlot}}^w \in \mathcal{A}_{\text{stateSlot}}^w$, and $a_{\text{yesNo}}^w \in \mathcal{A}_{\text{yesNo}}^w$. $\mathcal{A}_{\text{state}}^w$ consists of *state*$[w](s_u^w)$ and indicates the user said their goal *without identifying* which slot it corresponds to – for example, "London" or "10:00 AM". $\mathcal{A}_{\text{stateSlot}}^w$ consists of *stateSlot*$[w](s_u^w)$ and indicates the user said their goal *and identified* which slot it corresponds to – for example, "to London", "from London", "leaving at 10:00 AM", or "arriving at 10:00". Finally $\mathcal{A}_{\text{yesNo}}^w$ includes actions *yes* and *no*. The sets $\mathcal{A}_{\text{state}}^w$, $\mathcal{A}_{\text{stateSlot}}^w$, and $\mathcal{A}_{\text{yesNo}}^w$ each also contain *null*.

Next, the user action model $p(a'_u|s'_u, a)$ was extended to support this formulation. Each slot contains a slot-specific user model, conditioned on whether the machine is asking about this slot or another (i.e., *any* other) slot. To make the user action model as realistic as possible, real dialog data from the SACTI-1 corpus was employed [28]. The SACTI-1 corpus contains 144 human-human dialogs in the travel/tourist information domain using a "simulated ASR channel" [31]. The corpus contains a variety of word error rates, and the behaviors observed of the subjects in the corpus are broadly consistent with behaviors observed of a user and a computer using a real speech recognition system [28]. The corpus was segmented into a "training sub-corpus" and a "test sub-corpus," which are each composed of an equal number of dialogs, the same mix of word error rates, and disjoint subject sets. Wizard/User turn pairs were annotated with dialog acts such as *ask* and *confirm* for the wizard, and *yes*, *no*, *state* and *stateSlot* (as described above) for the user. One user model was then estimated from each sub-corpus, called the *training* user model and the *testing* user model.[10] Excerpts from these two models are shown in Table II. In the table, the dash character (—) indicates *null*.

A key property of real-world spoken dialog systems is that speech recognition errors are not confined to separate slots: for example, a time such as "ten a.m." may be mis-recognized as another time such as "eleven a.m.", or as a place such as "Tennessee" or an airline such as "Pan Am". To model this as closely as possible, the observation model was separated into a

[9] A further limitation is that, unlike PBVI, CSPBVI computes only the value of a conditional plan at each point, and not its value gradient. As a result, CSPBVI does not compute accurate boundaries between regions and relies on a nearest neighbor heuristic. However, a version of CSPBVI which maintained gradients was evaluated and obtained worse performance than the method presented here. It was observed that occasionally gradients would be poorly estimated which reduced plan quality and lowered returns, and it is believed this was caused by the non-linear mapping into summary space.

[10] The user models did not attempt to estimate statistics for "propositional content" of user utterances (such as "London") from the corpus, as it was assumed that the user would provide propositional content which was consistent with their goal. Rather the user model estimated the distribution of the "illocutionary force" of a user action for a given wizard action: for example, if the wizard asked about a particular slot, the user model estimated how often the user would provide a value for that slot, how often the user would include the name of the slot, and how often the user would provide a value for another slot.

TABLE II
SUMMARY OF USER MODEL PARAMETERS FOR THE MAXITRAVEL-W APPLICATION.

Machine Action		User Response (*to* slot only)		$P(a_u^{\text{to}\prime} \| s_u^{\text{to}\prime} = \text{London}, a)$	
Utterance	a	Utterance	$(a^{\text{to}\prime}_{\text{state}}, a^{\text{to}\prime}_{\text{stateSlot}}, a^{\text{to}\prime}_{\text{yesNo}})$	Training	Testing
"Where are you going to?"	*ask*[*to*]()	"London"	$(\textit{london}, -, -)$	0.521	0.532
		"To London"	$(-, \textit{london}, -)$	0.467	0.443
		—	$(-, -, -)$	0.013	0.025
"Where are you leaving from?"	*ask*[*from*]()	"To London"	$(-, \textit{london}, -)$	0.146	0.212
		—	$(-, -, -)$	0.855	0.788
"To London, is that right?"	*confirm*[*to*](*london*)	"Yes"	$(-, -, \textit{yes})$	0.782	0.806
		"Yes, London"	$(\textit{london}, -, \textit{yes})$	0.093	0.042
		"Yes, to London"	$(-, \textit{london}, \textit{yes})$	0.112	0.127
		—	$(-, -, -)$	0.013	0.025
"To Edinburgh, is that right?"	*confirm*[*to*](*edinburgh*)	"No"	$(-, -, \textit{no})$	0.782	0.806
		"No, London"	$(\textit{london}, -, \textit{no})$	0.093	0.042
		"No, to London"	$(-, \textit{london}, \textit{no})$	0.112	0.127
		—	$(-, -, -)$	0.013	0.025
"From Oxford, is that right?"	*confirm*[*from*](*oxford*)	"To London"	$(-, \textit{london}, -)$	0.245	0.522
		—	$(-, -, -)$	0.755	0.478

generation model and an *inference* model. An important goal in this work is to allow the user to say anything at any point in the dialog, and so it is assumed that the same recognition grammar is active throughout. To model this, the generation model makes concept confusions with a constant probability p_{err}, where a confusion substitutes a non-null user action component to any other component *in any slot*. For example, if one concept error is made, the user action "Yes, London" might be changed to "Frankfurt London" or even "Yes No". Since *null* is one type of user action, the generation model also simulates deletion errors – for example, "Yes, London" could be changed to "Yes", "London" or *null*. The model does not simulate insertion errors.

In addition, each observation component (such as "London" or "To Edinburgh" or "Yes") carries with it a *per-concept* confidence score c. Past work has found that confidence scores can be modelled with an exponential distribution [32], and here confidence scores for correct recognitions are sampled from $p_h(c)$ and incorrect recognitions are sampled from $p_h(1-c)$, where

$$p_h(c) \quad = \quad \frac{(he^{hc})}{e^h - 1}, \tag{10}$$

$c \in [0, 1]$, h is a constant that determines the "informativeness" of the confidence score. For example, for $h = 0$ the confidence score is random noise, and as h increases the confidence score becomes more reliable. To give an intuitive sense of the effect of h, consider a classifier which attempts to label utterances as *correct* or *incorrect* from a speech recognizer which makes errors 30% of the time ($p_{err} = 0.30$). When $h = 0$, the confidence score adds no information, and the minimum error rate possible for a classifier would be 30% (by marking all utterances as correct). At $h = 2$, the confidence score adds useful information, and the classifier's minimum error rate decreases to 23%. At $h = 5$, the minimum classification error rate is 8%, and as h approaches infinity, the minimum classification error rate approaches 0.

Ideally the observation inference model should express the probability of the entire observation given the entire user action $P(\tilde{a}'_u, c'|a_u)$, but this formulation would complicate belief monitoring significantly. Instead, the observation model estimates $P(\tilde{a}'_u, c'|a^{w\prime}_u)$ *separately* for each slot:

$$P(\tilde{a}'_u, c'|a^{w\prime}_u) \approx$$

$$\begin{cases} p_h(c'_i) \cdot (1 - p_{err}) & \text{if there exists an observation component } i \text{ in } \tilde{a}'_u \text{ with the same } \textit{type} \text{ as } a^{w\prime}_u, \\ \frac{p_{err}}{|\mathcal{A}^w_u| - 1} & \text{otherwise,} \end{cases} \tag{11}$$

where example *types* include "place names", "dates", or "boolean" (yes/no).

The reward function provided a large positive reward $(+12.5 \cdot W)$ for taking a correct submit action; a large penalty $(-12.5 \cdot W)$ for taking an incorrect submit action; and a host of smaller penalties ranging from -1 to -3 depending on the appropriateness of information gathering actions and the grounding state given in s_d.

A CSPBVI-based dialog manager requires a heuristic which chooses among actions nominated by each slot. For the MAXITRAVEL-W application, this heuristic first looks for an *ask* action by considering the slots in order; if it does not find one, it then looks for a *confirm* action again considering the slots in order; and if it does not find one then all slots must have nominated the *submit* action, which is selected.

The CSPBVI optimization procedure takes a set of parameters N, K, and T where N is the number of belief points in summary space sampled for each slot; K is the number of successor belief points sampled for each summary action taken at each belief point; and T is the planning horizon. Experimentation found that no gains in performance were achieved for values beyond $N = 100$, $K = 50$, and $T = 50$, and these values were used for all experiments, below.

As an illustration, Figure 5 shows a conversation with a 2-slot version of MAXITRAVEL-W with a typical concept error rate ($p_{err} = 0.30$) and a somewhat informative confidence score ($h = 2$). This figure shows only the user goal component – the dialog history component has been omitted due to space limitations. In this example, the user is trying to buy a ticket from London to Cambridge.

Initially belief mass is spread over all user goals evenly, and at the beginning of the conversation, the summary belief state for each slot is passed to its respective policy, and both the *from* slot and the *to* slot nominate the *ask* action. The heuristic examines these two actions and selects the *ask*[*from*]() action (M1). The user's response (U1) of "London" is mis-recognized as "Leeds" with relatively high confidence (0.67), causing a large shift in belief mass toward *leeds* in the *from* slot, and no change to the belief state of the *to* slot. For the second machine action (M2), the *from* slot nominates the *confirm* action and the *to* slot nominates the *ask* action. The heuristic examines these and selects the *ask*[*to*]() action (M2). The user's response (U2) of "To Cambridge" is again mis-recognized, this time as "From Oxford", with moderate confidence (0.45). However, since the user model indicates this is a relatively unlikely user action, and because the confidence score is somewhat low, this results in only a moderate shift of belief mass toward *oxford* in the *from* slot (and again no change to the *to* slot). For the third machine action (M3), the *from* slot again nominates the *confirm* action and the *to* slot nominates the *ask* action; the heuristic examines these and again selects the *ask*[*to*]() action (M3).

The user's response to M3 (U3) of "To Cambridge" is correctly recognized with very high confidence (0.97). This causes a massive shift of belief mass toward *cambridge* in the *to* slot (and no change to the *from* slot). For the fourth machine action (M4), the *from* slot is in the same belief state and again nominates the *confirm* action. The *to* slot is now in a belief state with high certainty, and nominates the *submit* action. The heuristic examines these and selects the *confirm*[*from*](*leeds*) action (M2). To this the user responds "No, from London" (U4) but this is mis-recognized as "No, to Edinburgh". The *no* portion of the response is recognized with high confidence (0.78) and is also highly predicted by the user model, so as a result a great deal of belief mass moves away from *leeds* in the *from* slot. However, the "to Edinburgh" portion is predicted as highly unlikely by the user model, and is also recognized with very low confidence (0.13), and has virtually no effect on the *to* slot. For the fifth machine action (M5), the *from* slot is now in a belief state with relatively low certainty, and nominates the *ask* action. The *to* slot is still in a state of very high certainty and again nominates the *submit* action. Between these the heuristic selects the "ask[from]()" action (M5), and the user's response of "From London to Cambridge" (U5) is correctly understood. The "From London" portion is recognized with very high confidence (0.95) and causes a massive shift in belief mass; the "to Cambridge" portion is recognized with relatively low confidence (0.33), and as a result has only a marginal effect on the *to* slot. For the sixth machine action, both slots are in a state of high certainty and both nominate the *submit* action; the heuristic chooses $a = \textit{submit}[\textit{all}](s_u^{\text{from}} = \textit{london}, s_u^{\text{to}} = \textit{cambridge})$ and the dialog is completed successfully.

B. Results

First CSPBVI was compared to a variety of fully-observable Markov decision processes (MDPs). Past work has shown that MDPs are an effective technique for building good dialog managers [33], [34], [32]. Like POMDPs, MDPs perform planning, but unlike POMDPs they maintain a single discrete state. POMDPs have previously been shown to outperform MDPs on very small dialog problems [1], [2], [4], [6], and here it will be verified that the assumptions made by CSPBVI do not compromise the ability of the POMDP to outperform an MDP baseline.

The first comparison sought to determine the value of performing belief monitoring (i.e., maintaining multiple hypotheses for the dialog state). In this experiment, no confidence score information is used (i.e., $h = 0$). Two MDP baselines were created. Both MDPs used a state estimator which received the speech recognition hypothesis $\tilde{a}_u$ as input and tracked whether each slot was *not-stated*, *unconfirmed*, or *confirmed* using basic grounding rules. The first MDP baseline, "MDP-Full", formed its state space as the cross-product of all MDP slot-states and the second MDP baseline, "MDP-Composite", estimated a separate MDP policy for each slot and used the same heuristic to choose actions at runtime as CSPBVI. In all other respects the simulation environment for the MDPs and CSPBVI were identical (e.g., the MDP action set included the same actions as in the CSPBVI action set). Both MDP baselines were trained using Q-learning [35]. A variety of learning parameters were explored and the best-performing set were selected: 100,000 training dialogs, initial Q values set to 0, exploration parameter $\epsilon = 0.2$, and learning rate $\alpha = 1/m$, where m is the number of visits to the $Q(s, a)$ being updated. This experiment was repeated for $w = 1 \ldots 5$ at a variety of error rates using the same optimization parameters. Results are shown in Figure 6. When no recognition errors are made (i.e., $p_{err} = 0.00$), the POMDP and MDPs perform identically but when concept recognition errors are made (i.e., $p_{err} > 0$), the POMDP outperforms the MDP. As the number of slots increases, average return declines slightly for all techniques, because eliciting values for more slots results in longer dialogs. Past work has shown that POMDPs cope with conflicting evidence better than MDPs; the findings here agree and suggest that the approximations made by CSPBVI do not compromise this crucial characteristic of the POMDP approach.

Next, the effect of confidence score was investigated by varying h. The concept error rate was set to $p_{err} = 0.30$. For the MDP-Composite baseline, a "confidence bucket" feature was added to the MDP state space representing "high" and "low" observed confidence scores. A variety of confidence thresholds were explored and it was found that using a threshold of 0.5 produced optimal results for the MDP.[11] Results are shown in Figure 7. As the confidence score becomes more informative (i.e., as h increases), performance increases for both the CSPBVI and MDP policies. Past work has found that

[11]The other techniques considered included dividing the probability mass of the confidence score c evenly between buckets, and dividing the range of error rates equally (e.g., for two buckets, setting a threshold such that $p(\text{observation is correct}|\text{confidence score}) = 0.5$).

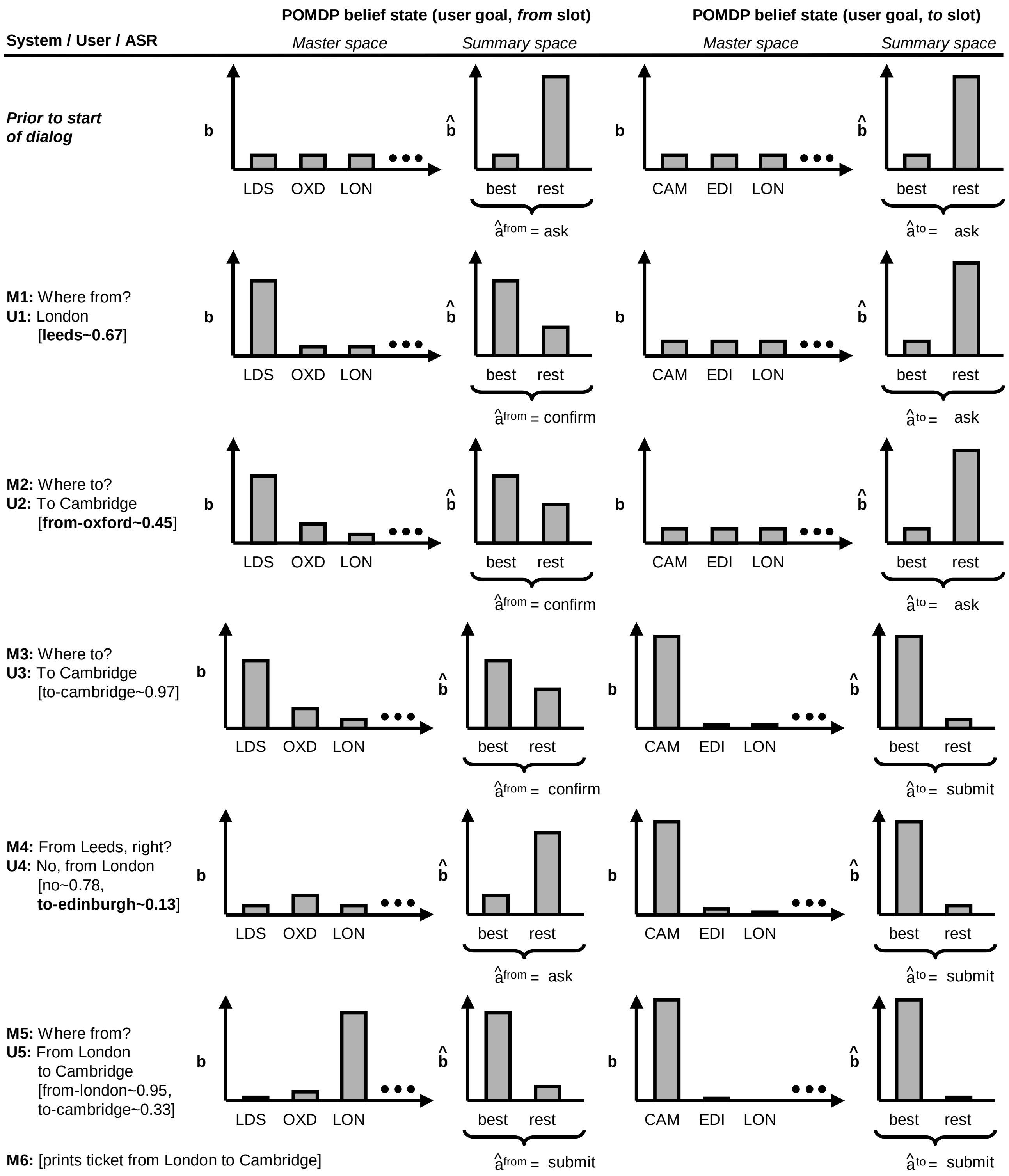

Fig. 5. Example conversation with the CSPBVI dialog manager. LDS stands for Leeds, OXD for Oxford, LON for London, CAM for Cambridge, and EDI for Edinburgh. a^{from} and a^{to} indicate actions nominated by each slot. Numbers indicate confidence scores; boldface highlights concept recognition errors. Only the user goal component of the POMDP state has been shown – the dialog history and user action components have been omitted due to space limitations. As described in Algorithm 4, the policy for each slot nominates a summary action such as $\hat{a}^{\text{from}} = \text{confirm}$ or $\hat{a}^{\text{to}} = \text{submit}$. These actions are mapped to master space using Algorithm 2 which appends the most likely user goal for the slot, converting (for example) $\hat{a}^{\text{from}} = \text{confirm}$ into $a = \text{confirm[from](leeds)}$.

POMDPs make better use of confidence score information than MDPs on very small dialog problems [5], and Figure 7 indicates that this trend continues at scale.

CSPBVI optimization was then compared to two handcrafted dialog managers, HC1 and HC2. HC1 and HC2 both use the same state estimator as the “MDP-Composite” baseline. Both HC1 and HC2 take the *ask* action for *not-stated* slots, and the *submit* action for *confirmed* slots. For *unconfirmed* slots, HC1

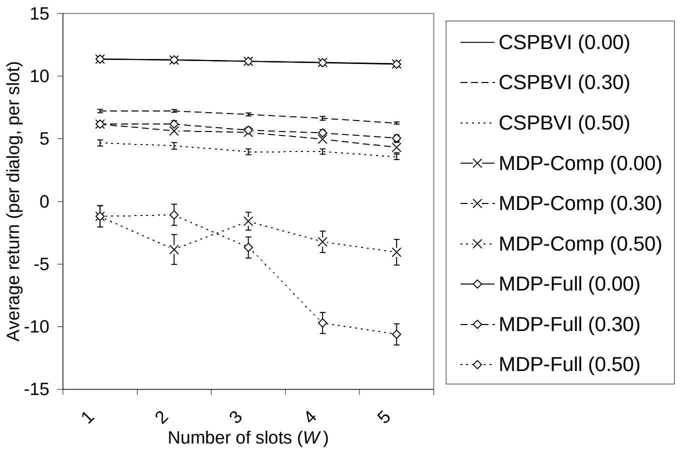

Fig. 6. Average return vs. number of slots (W) for CSPBVI and two MDP baselines at various concept error rates. At the 0.00 error rate, the solid lines for CSPBVI MDP-Comp and MDP-Full are equal and are overlaid.

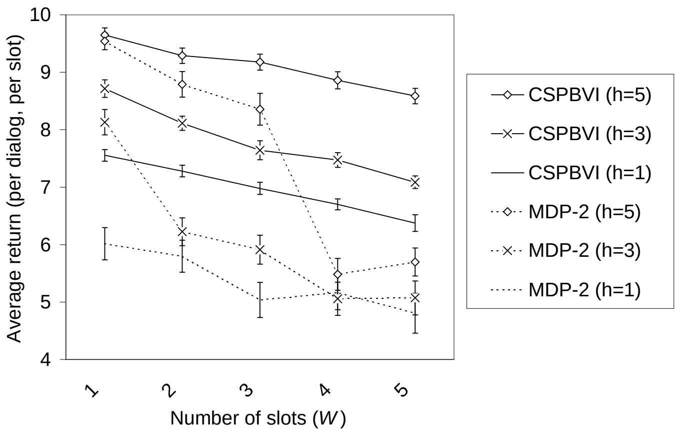

Fig. 7. Average return vs. number of slots (W) for CSPBVI and MDP-Composite-2 baseline for and for concept error rate $p_{err} = 0.30$ at various levels of confidence score reliability (h).

takes the *confirm* action and HC2 takes the *ask* action. HC1 and HC2 were evaluated by running 10,000 simulated dialogs for various number of slots and error rates. Results are shown in Figure 8.

CSPBVI outperforms both handcrafted controllers at all error rates. As the number of slots increases, the reward gained per slot decreases, but at higher error rates (i.e., $p_{err} = 0.50$) this decline is precipitous for the handcrafted controllers but gradual for the POMDP. One reason for this is that the POMDP is making use of a user model and taking proper account of less likely observations, but the handcrafted policies place equal trust in all observations. As dialogs become longer, the simulated user provides less-reliable information about *other* slots more times in each dialog, causing the performance of handcrafted policies to degrade.

To this point, all experiments have been optimized and evaluated on the same user model (*training* user model in Table II). In practice, a user model will be a noisy estimate of real user behavior, and experiments so far have not addressed what effect this deviation might have on performance. To model this, a 5-slot MAXITRAVEL-W was trained using the

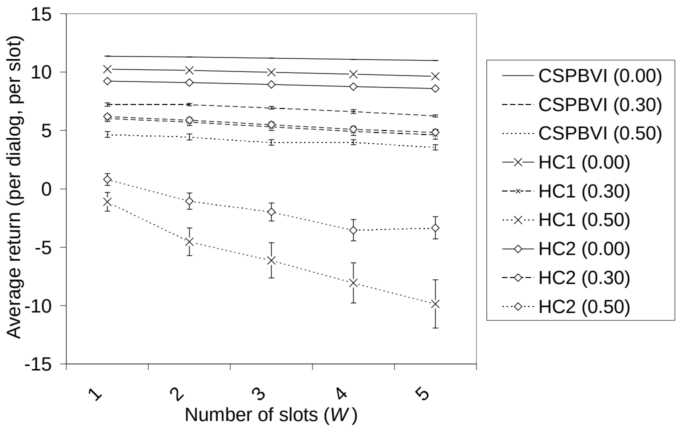

Fig. 8. Average return vs. number of slots for CSPBVI and handcrafted baselines at various concept error rates.

training user model and evaluated using the *testing* user model described above. Results are shown in Figure 9. As speech recognition errors increase, the average reward per turn decreases as expected, and in general performance on the test user model is less than but very close to the training user model, implying that the method is reasonably robust to variations in patterns of user behavior or estimation errors in the user model.

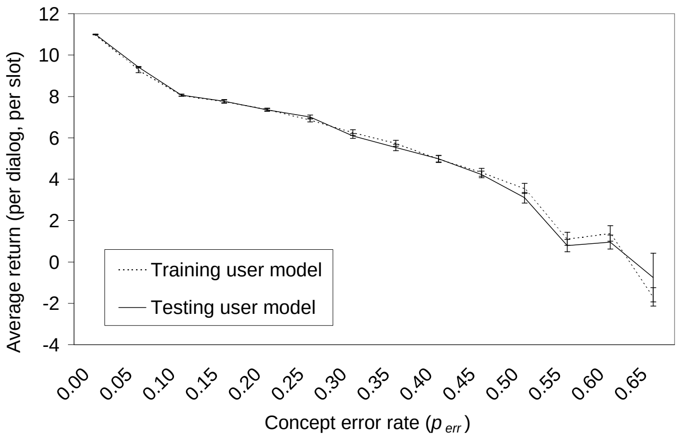

Fig. 9. Average return vs. concept error rate for training and testing user models.

Finally, the scalability of CSPBVI was compared to direct optimization. A simplified, one-slot version of the MAXITRAVEL-W application was created, with a concept error rate (p_{err}) of 0.30 and no confidence score information ($h = 0$). The number of slot values (M) was initially set to 3. CSPBVI was run on this application with $N = 100$ belief points (in summary space), and as a baseline, PBVI (as implemented in *Perseus* [14]) was also run on this application with $N = 1000$ belief points (in master space). This process was repeated for values of M (i.e., distinct number of slot values) from 3 to 5000.

Figure 10 shows M vs. average return for CSPBVI and the PBVI baseline. For small problems, i.e., lower values of M, CSPBVI performs equivalently to the PBVI baseline, but for larger problems, CSPBVI outperforms PBVI by an increasing margin until $M = 100$ at which point PBVI was not able to find policies. Moreover, the CSPBVI policies were computed using 90% fewer belief points than the baseline; i.e., CSPBVI policies scale to large problems and are much more compact.

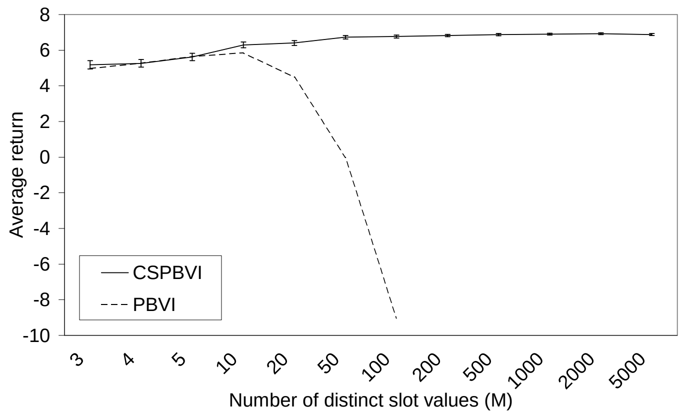

Fig. 10. Average return vs. number of distinct slot values (M) for a simplified 1-slot dialog problem.

It is interesting to note that as M increases, the performance of the summary POMDP method appears to increase toward an asymptote. This trend is due to the fact that all confusions are equally likely in this model. For a given error rate, the more concepts in the model, the less likely consistent confusions are. Thus, having more concepts actually helps the policy identify spurious evidence over the course of a dialog. In practice of course the concept error rate p_{err} may increase as concepts are added.

VI. Conclusions

CSPBVI enables slot-filling dialog systems cast as SDS-POMDPs to be scaled to handle many slots. In dialog simulation, the scalability gained with localized planning maintains performance gains over baseline techniques while tolerating errors in user model estimation. Although CSPBVI makes several important assumptions – i.e., sufficient coverage of belief space, lack of value gradient estimation, Markov assumption violations, and localized planning – in practice CSPBVI outperforms MDP and hand-crafted dialog manager baselines while scaling to problems beyond the reach of other POMDP optimization techniques. Now that realistically-sized systems can be created, our next step is to conduct trials of CSPBVI-based dialog managers with real users.

CSPBVI was created for the dialog management domain, but it could also be applied to similarly structured problems in other domains. For example, consider a fault remediation task consisting of a network with many components. Tests may be run which give an indication of faulty component, and the ultimate goal is to decide which component to replace. In this scenario, the identity of the particular component is less important than how likely it is to be the culprit given the evidence. It would be interesting to try applying CSPBVI to problems like this outside of the dialog domain.

Appendix
Algorithm listings

Algorithm 5: Sampling stage of CSPBVI.

```
Input: $\mathfrak{P}_{SDS}$, $\epsilon$, $N$, $K$, $W$
Output: $\{\hat{b}_{w,n}\}$,$\{\hat{r}^{\hat{a},k}_{w,n}\}$, $\{l(w,n,\hat{a},k)\}$
// Iterate over each slot $w$.
for $w \leftarrow 1$ to $W$ do
    // First, sample points
    // using a random policy
    $s \leftarrow$ sampleFromDistribution($b$)
    $n \leftarrow 1$
    $b \leftarrow b_0$
    $\hat{b}_{w,n} \leftarrow$ bToSummary($b, w$)
    $(\hat{r}^{\hat{a},k}_{w,n}, \hat{b}^{\hat{a},k}_{w,n}) \leftarrow$ samplePoint($\mathfrak{P}_{SDS}, b, w, K$)
    while $n < N$ do
        // Take a random action and
        // compute new belief state.
        $\hat{a} \leftarrow$ randomElement($|\hat{\mathcal{A}}|$)
        $\hat{w} \leftarrow$ randomElement($\{$this, other$\}$)
        $a \leftarrow$ aToMaster($\hat{a}, \hat{w}, w, b$)
        $s' \leftarrow$ sampleFromDistribution($P(s'|s,a)$)
        $o' \leftarrow$ sampleFromDistribution($P(o'|s',a)$)
        $b \leftarrow SE(b, a, o')$
        $\hat{b} \leftarrow$ bToSummary($b, w$)
        // If this is a (sufficiently)
        // new point, add it to $\hat{\mathcal{B}}$.
        if $\min_{i \in [1,n]} |\hat{b}_{w,i} - \hat{b}| > \epsilon$ then
            $n \leftarrow (n+1)$
            $\hat{b}_{w,n} \leftarrow \hat{b}$
            $(\hat{r}^{\hat{a},k}_{w,n}, \hat{b}^{\hat{a},k}_{w,n}) \leftarrow$ samplePoint($\mathfrak{P}_{SDS}, b, w, K$)
        $s \leftarrow s'$
    // Second, sample corners $\hat{s}$
    // of summary space
    foreach $\hat{s} \in \hat{\mathcal{S}}_w$ do
        foreach $\tilde{s} \in \hat{\mathcal{S}}_w$ do
            $\hat{b}(\tilde{s}) \leftarrow 0$
        $\hat{b}(\hat{s}) \leftarrow 1$
        if $\min_{i \in [1,n]} |\hat{b}_{w,i} - \hat{b}| > \epsilon$ then
            $n \leftarrow (n+1);\ N \leftarrow n$
            $b \leftarrow$ sampleCorner($b_0, \hat{s}$)
            $\hat{b}_{w,n} \leftarrow$ bToSummary($b, w$)
            $(\hat{r}^{\hat{a},k}_{w,n}, \hat{b}^{\hat{a},k}_{w,n}) \leftarrow$ samplePoint($\mathfrak{P}_{SDS}, b, w, K$)
    // Third, find index $l(w,n,\hat{a},k)$ of
    // closest point to $b^{\hat{a},k}_{w,n}$
    for $n \leftarrow 1$ to $N$ do
        foreach $\hat{a} \in \hat{\mathcal{A}}^w$ do
            for $k \leftarrow 1$ to $K$ do
                $l(w,n,\hat{a},k) \leftarrow \arg\min_{\tilde{n}} |b^{a,k}_{w,n} - b_{w,\tilde{n}}|$
```

Algorithm 6: Function *samplePoint* used by CSPBVI sampling (Algorithm 5).

```
   Input: $\mathfrak{P}_{SDS}$, $b$, $w$, $K$
   Output: $\{\hat{b}^{\hat{a},k}\}$, $\{\hat{r}^{\hat{a},k}\}$
1  foreach $\hat{a} \in \hat{\mathcal{A}}^w$ do
2      foreach $\hat{w} \in \hat{\mathcal{W}}$ do
3          for $k \leftarrow 1$ to $K$ do
               // Sample a (possibly) new
               // master state $\tilde{s}$.
4              $\tilde{s} \leftarrow$ sampleFromDistribution$(b)$
5              $a \leftarrow$ aToMaster$(\hat{a}, \hat{w}, w, b)$
6              $s' \leftarrow$ sampleFromDistribution$(P(s'|\tilde{s}, a))$
7              $o' \leftarrow$ sampleFromDistribution$(P(o'|s', a))$
               // Compute successor $b'$ and
               // save $\hat{r}^{\hat{a}[\hat{w}],k}$ and $\hat{b}^{\hat{a}[\hat{w}],k}$.
8              $b' \leftarrow SE(b, a, o')$
9              $\hat{r}^{\hat{a}[\hat{w}],k} \leftarrow r_w(\tilde{s}, a)$
10             $\hat{b}^{\hat{a}[\hat{w}],k} \leftarrow$ bToSummary$(b', w)$
```

Algorithm 7: Function *sampleCorner*.

```
   Input: $b$, $\hat{s}$
   Output: $b$
1  $s_u^* \leftarrow \arg\max_{s_u} b(s_u)$
2  if $\hat{s}_u =$ best then
       // When $\hat{s}_u =$ best, best guess for
       // user's goal is correct:
       // set $s_u$ to this guess and $b(s_u)$
       // to that corner.
3      foreach $s_u \in \mathcal{S}_u$ do
4          $b(s_u) \leftarrow 0$
5      $b(s_u^*) \leftarrow 1$
6  else
       // When $\hat{s}_u =$ rest, best guess for
       // user's goal is NOT correct:
       // set $b(s_u^*)$ to zero and
       // renormalize, then
       // sample $s_u$ from this renormalized
       // belief state.
7      $b(s_u^*) \leftarrow 0$
8      norm $\leftarrow \sum_{s_u} b(s_u)$
9      foreach $s_u \in \mathcal{S}_u$ do
10         $b(s_u) \leftarrow \frac{b(s_u)}{\text{norm}}$
   // Copy $\hat{b}(s_d)$ directly into $b(s_d)$.
11 foreach $s_d \in \mathcal{S}_d$ do
12     $b(s_d) \leftarrow \hat{b}(s_d)$
```

Acknowledgment

This work has been supported in part by the EU FP6 Talk Project.

References

[1] N. Roy, J. Pineau, and S. Thrun, "Spoken dialog management for robots," in *Proc Association for Computational Linguistics (ACL), Hong Kong*, 2000.

[2] B. Zhang, Q. Cai, J. Mao, E. Chang, and B. Guo, "Spoken dialogue management as planning and acting under uncertainty," in *Proc Eurospeech, Aalborg, Denmark*, 2001.

[3] B. Zhang, Q. Cai, J. Mao, and B. Guo, "Planning and acting under uncertainty: A new model for spoken dialogue system," in *Proc Conference in Uncertainty in Artificial Intelligence (UAI)*, 2001, pp. 572–579.

[4] J. Williams, P. Poupart, and S. Young, "Factored partially observable Markov decision processes for dialogue management," in *Proc Workshop on Knowledge and Reasoning in Practical Dialog Systems, Intl Joint Conf on Artificial Intelligence (IJCAI), Edinburgh*, 2005.

[5] ——, "Partially observable Markov decision processes with continuous observations for dialogue management," in *Proc SIGdial Workshop on Discourse and Dialogue, Lisbon*, 2005.

[6] J. Williams and S. Young, "Partially observable markov decision processes for spoken dialog systems," *Computer Speech and Language*, vol. 21, no. 2, pp. 393–422, 2007.

[7] A. Drake, "Observation of a markov process through a noisy channel," Ph.D. dissertation, Massachusetts Institute of Technology, 1962.

[8] K. Astrom, "Optimal control of markov decision processes with incomplete state estimation," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, 1965.

[9] E. Sondik, "The optimal control of partially observable Markov decision processes," Ph.D. dissertation, Stanford University, 1971.

[10] R. Smallwood and E. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations Research*, vol. 21, pp. 1071–1088, 1973.

[11] G. Monahan, "A survey of partially observable Markov decision processes: Theory, models, and algorithms," *Management Science*, vol. 28, no. 1, pp. 1–16, 1982.

[12] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, 1998.

[13] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: an anytime algorithm for POMDPs," in *Proc Intl Joint Conf on Artificial Intelligence (IJCAI), Acapulco, Mexico*, 2003.

[14] M. Spaan and N. Vlassis, "Perseus: randomized point-based value iteration for POMDPs," *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, 2005.

[15] S. Larsson and D. Traum, "Information state and dialogue management in the TRINDI dialogue move engine toolkit," *Natural Language Engineering*, vol. 5, no. 3/4, pp. 323–340, 2000.

[16] M. F. McTear, *Spoken dialogue technology: toward the conversational user interface*. Springer Verlag, 2004.

[17] T. Paek and E. Horvitz, "Uncertainty, utility, and misunderstanding: A decision-theoretic perspective on grounding in conversational systems," in *AAAI Fall Symposium on Psychological Models of Communication, North Falmouth, MA, USA*, 1999, pp. 85–92.

[18] E. Horvitz and T. Paek, "DeepListener: Harnessing expected utility to guide clarification dialog in spoken language systems," in *Proc Intl Conf on Spoken Language Processing (ICSLP), Beijing*, 2000.

[19] T. Paek and E. Horvitz, "Conversation as action under uncertainty," in *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI), Stanford, CA*, 2000.

[20] C. Boutilier and D. Poole, "Computing optimal policies for partially observable decision processes using compact representations," in *Proc Thirteenth National Conference on Artificial Intelligence, Portland, OR, USA*, 1996, pp. 1168–1175.

[21] E. Hansen and Z. Feng, "Dynamic programming for pomdps using a factored state representation," in *Proc Fifth International Conference on AI Planning Systems, Breckenridge, CO, USA*, 2000, pp. 130–139.

[22] N. Roy and G. Gordon, "Exponential family PCA for belief compression in POMDPs," in *Proc Advances in Neural Information Processing Systems (NIPS), Vancouver, BC, Canada*, 2002, pp. 1635–1642.

[23] P. Poupart and C. Boutilier, "VDCBPI: an approximate scalable algorithm for large scale POMDPs," in *Proc Advances in Neural Information Processing Systems 17 (NIPS), Vancouver, Canada*, 2004, pp. 1081–1088.

[24] C. Guestrin, D. Koller, and R. Parr, "Solving factored POMDPs with linear value functions," in *Proc Intl Joint Conf on Artificial Intelligence (IJCAI) Workshop on Planning under Uncertainty and Incomplete Information, Seattle*, 2001.

[25] J. Pineau, "Tractable planning under uncertainty: Exploiting structure," Ph.D. dissertation, Carnegie Mellon University, 2004.

[26] J. Williams and S. Young, "Scaling up POMDPs for dialog management: The "summary POMDP" method," in *Proc IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), San Juan, Puerto Rico, USA*, 2005.

[27] ——, “Scaling POMDPs for dialog management with composite summary point-based value iteration (CSPBVI),” in *Proc American Association for Artificial Intelligence (AAAI) Workshop on Statistical and Empirical Approaches for Spoken Dialogue Systems*, 2006.
[28] ——, “Characterizing task-oriented dialog using a simulated ASR channel,” in *Proc Intl Conf on Speech and Language Processing (ICSLP), Jeju, Korea*, 2004.
[29] R. Sutton and A. Barto, *Reinforcement Learning: an Introduction*. MIT Press, 1998.
[30] O. Lemon, K. Georgila, and J. Henderson, “Evaluating effectiveness and portability of reinforcement learned dialogue strategies with real users: the TALK TownInfo evaluation,” in *Proc IEEE/ACL Workshop on Spoken Language Technology (SLT), Aruba*, 2006.
[31] M. Stuttle, J. Williams, and S. Young, “A framework for Wizard-of-Oz experiments with a simulated ASR channel,” in *Proc Intl Conf on Speech and Language Processing (ICSLP), Jeju, Korea*, 2004.
[32] O. Pietquin, “A framework for unsupervised learning of dialogue strategies,” Ph.D. dissertation, Faculty of Engineering, Mons (TCTS Lab), Belgium, 2004.
[33] E. Levin, R. Pieraccini, and W. Eckert, “A stochastic model of human-machine interaction for learning dialogue strategies,” *IEEE Trans. Speech Audio Processing*, vol. 8, no. 1, pp. 11–23, 2000.
[34] S. Singh, D. Litman, M. Kearns, and M. Walker, “Optimizing dialogue management with reinforcement leaning: experiments with the NJFun system,” *Journal of Artificial Intelligence*, vol. 16, pp. 105–133, 2002.
[35] C. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, Cambridge University, 1989.

Jason D. Williams is a Senior Member of Technical Staff at AT&T Labs – Research in Florham Park, New Jersey, USA. He received a BSE in Electrical Engineering from Princeton University in 1998, and at Cambridge University he received an M Phil in Computer Speech and Language Processing in 1999 under a Churchill Scholarship and a Ph D in Information Engineering in 2006 under a Gates Scholarship. His main research interests are dialog management, the design of spoken language systems, and planning under uncertainty. He has previously held positions at Tellme Networks, Edify Corporation (now Intervoice), and McKinsey & Company’s Business Technology Office.

PLACE
PHOTO
HERE

Steve Young received a BA in Electrical Science Tripos from Cambridge University in 1973 and a PhD in 1977. He was then a lecturer at UMIST and Cambridge before being elected to a Chair in Information Engineering at Cambridge in 1995 where he is currently Head of the Information Engineering Division. Interleaved with his academic appointments, he has also held industrial positions with GEC, Entropic and Microsoft. His main research interests lie in the area of spoken language systems including speech recognition, speech synthesis and dialogue management. He was Editor of Computer Speech and Language from 1993 to 2004. He is a Fellow of the Royal Academy of Engineering, the Institution of Electrical Engineers and the RSA. He is a member of the British Computer Society and the Institute of Electrical and Electronics Engineers. In 2004, he was a recipient of an IEEE Signal Processing Society Technical Achievement Award.