A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies

Esther Levin, *Member, IEEE*, Roberto Pieraccini, *Member, IEEE*, and Wieland Eckert, *Member, IEEE*

***Abstract*—In this paper, we propose a quantitative model for dialog systems that can be used for learning the dialog strategy. We claim that the problem of dialog design can be formalized as an optimization problem with an objective function reflecting different dialog dimensions relevant for a given application. We also show that any dialog system can be formally described as a sequential decision process in terms of its state space, action set, and strategy. With additional assumptions about the state transition probabilities and cost assignment, a dialog system can be mapped to a stochastic model known as *Markov decision process* (MDP). A variety of data driven algorithms for finding the optimal strategy (i.e., the one that optimizes the criterion) is available within the MDP framework, based on reinforcement learning. For an effective use of the available training data we propose a combination of supervised and reinforcement learning: the supervised learning is used to estimate a model of the user, i.e., the MDP parameters that quantify the user's behavior. Then a reinforcement learning algorithm is used to estimate the optimal strategy while the system interacts with the simulated user. This approach is tested for learning the strategy in an air travel information system (ATIS) task. The experimental results we present in this paper show that it is indeed possible to find a simple criterion, a state space representation, and a simulated user parameterization in order to automatically learn a relatively complex dialog behavior, similar to one that was heuristically designed by several research groups.**

***Index Terms*—Dialog systems, Markov decision process, reinforcement learning, sequential decision process, speech, spoken language systems.**

I. Introduction

HUMAN-MACHINE interactions are ubiquitous in today's world. We are not surprised anymore when we interact and struggle with menu-driven touch tone or voice telephone systems, when we engage in an interaction with our PC through dialog boxes, or make a transaction by interacting with ATM machines or Web browsers. New applications such as those in [7], and [32] approach the frontier of natural language dialog to automate services that are currently performed by human operators. In the natural language and artificial intelligence (AI) research communities there are more complex examples of dialog systems that make use of natural ways of communication like speech and language [3]–[5], [12]–[14], [19], [23], [24]. All these are examples of man-machine dialog systems that notwithstanding their preferred way of interaction with the human (touch tone, keyboard, natural language, etc.) share a common principle: a strategy aimed at the satisfaction of the goal that justifies their existence.

Since there are no commonly accepted methodologies for building a dialog system, today dialog design is more art than engineering or science. In general, a dialog system consists of a more or less structured program, based on reasonable rules dictated both by the knowledge of the application and by continuous experimentation with real users. Sometimes the knowledge of the application takes the form of a plan [19], [33] that decomposes the original goal of the dialog system into a hierarchy of subgoals. However, the design principles are purely based on heuristics and trial-and-error. The problem with such a procedure is that generally it is hard to predict all the possible situations the system might encounter in real scenarios, hence a system can reach good performance only at the expense of an extended experimentation. Moreover, when a new application is developed, often the whole design process has to be started from scratch.

The research community in the field of spoken language systems is well aware of the benefits derived by the introduction of well-founded mathematical problem formalization. For instance, the tremendous progress made in speech recognition during the last decade is mainly due to the introduction and use of stochastic modeling of both acoustic [30] and linguistic phenomena [16]. Also, recent advances in language understanding for spontaneous speech are due to the introduction of different stochastic models [11], [14], [15], [21], [26]. The strength of the stochastic approach is in the fact that it naturally takes advantage of large amounts of data. In fact no one doubts the importance of data corpora in the spoken language community ("No data like more data"[1] has been the speech recognition motto in the past ten years). Data is used for estimating model parameters in order to optimize the performance of the system. Usually the more data the system is trained with, the higher the resulting performance on new data is. Moreover, since the training is done automatically and requires little or no supervision, new applications can be developed with the only cost of collecting new data, if at all needed.

These advantages of the data driven approach motivated the dialog community to start looking at them also for dialog design.

Manuscript received December 23, 1998; revised August 3, 1999. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. James R. Glass.

E. Levin is with AT&T Laboratories-Research, Shannon Laboratories, Florham Park, NJ 07932 USA (e-mail: esther@research.att.com).

R. Pieraccini was with AT&T Laboratories-Research, Shannon Laboratories, Florham Park, NJ 07932 USA. He is now with SpeechWorks International, Inc., New York, NY 10004 USA (e-mail: roberto.pieraccini@speechworks.com).

W. Eckert was with AT&T Laboratories-Research, Shannon Laboratories, Florham Park, NJ 07932 USA. He is now with Lucent Technologies, Nuernberg, Germany (e-mail: weckert@lucent.com).

Publisher Item Identifier S 1063-6676(00)00296-0.

[1]This sentence has been attributed to Bob Mercer, former researcher of the IBM speech group.

However it is not clear yet how to use data for that purpose. Of course, dialog designers do look at data in order to build the rules that guide the dialog, and later to improve its performance. But designers can analyze only a relatively small quantity of data, and there is not a clear understanding of which measures can be derived automatically from the corpus in order to automate the design procedure.

The main problem of supervised learning of dialog from corpus originates from the dynamic nature of dialog itself. Not only is it not clear how to learn a dialog strategy, but also a dialog system cannot be evaluated against a fixed corpus: if at any point during the evaluation the system deviates from the dialog in the corpus (for example because of an error or a change in the dialog design), the dialog will take a different course that cannot be predicted. In fact, even if we manage to learn a dialog strategy that approximates the one in the corpus with high accuracy and only few deviations, as a result of these deviations the dialog system can get into an unexplored portion of the state space that was not represented in the corpus. Moreover, the dialog strategies adopted by humans are not necessarily those best suited for automatic systems.

In this paper, we propose to use a combination of supervised and reinforcement learning. The dialog system will learn an optimal strategy using reinforcement learning while interacting with users. Reinforcement learning allows one to actively explore the state space of the system, while using delayed feedback to update the values of all states traversed in the current interaction. Although it is possible to use reinforcement learning with real users, it is often impractical. Here we propose to use supervised learning from a corpus of dialogs in order to estimate the parameters of a simulated user that is a stochastic model used in a generative mode. The simulated user then interacts with the dialog system while it uses reinforcement learning.

II. Dialogue Design as an Optimization Problem

In general, a dialog system is a machine that tries to achieve an application goal in an efficient way through a series of interactions with the user. By quantifying the concepts of achievement of an application goal and efficiency, we can state the problem of dialog design as optimization of an objective C:

$$C = \sum W_i \langle C_i \rangle \tag{1}$$

where the terms $\langle C_i \rangle$ are the expected costs for different dialog dimensions reflecting the distance to the achievement of the application goal and the efficiency of the interaction, and the weights W_i determine the tradeoff among the costs. Some of these dimensions can be measured directly by the system, like dialog duration, cost of internal processing, cost of accessing external databases or other resources and cost of ineffectiveness (e.g., number of errors the system made due to poor speech recognition); others quantify such abstract dimensions as user satisfaction (e.g., a simple happy/not happy-with-the-system feedback from the user at the end of dialog, number of users hanging up before the completion of the dialog goal, etc.). The work described in [9] relates user satisfaction to a linear combination of directly measurable quantities. Using the optimality criterion expressed by (1) we can evaluate system performance simply by running several dialog sessions and computing the average cost. Since the actions taken by the system may affect some or all the terms of the objective function, the optimal dialog strategy is a result of a correct trade off between them, as illustrated by the examples in this paper. The problem of dialog design that we address in this paper is that of finding the optimal strategy automatically.

Recently, several data-driven optimization approaches were attempted toward dialog design. For example, in [35], clarification questions are generated based on task description, avoiding the situation of asking the user a question that does not contribute to current information exchange. In [36], the next clarification attribute is computed from the retrieved data in order to minimize the expected number of clarification interactions. In both cases, a local optimization is performed in the particular state of the dialog where clarification is needed. In the approach proposed in this paper, the optimization is global over the whole dialog session.

We will illustrate the concepts introduced in this paper with a tutorial example of "Day-and-Month Dialog," where the goal of the system is to get the *correct* day and month values from the user through the *shortest* possible interaction.

From the definition of the application given above, it is straightforward to describe the objective function as the sum of three terms:

$$C = W_i \langle N_i \rangle + W_e \langle N_e \rangle + W_f \langle N_f \rangle. \tag{2}$$

The first term is the expected duration of the dialog (N_i being the number of interactions). The second term corresponds to the expected number of errors N_e in the obtained values (ranging from zero to two); and the third measures the expected distance N_f from achieving our application goal (this distance is zero for a complete date, one if either day or month value is missing, and two if both are incomplete).

We formalize a dialog system by describing it as a sequential decision process in terms of its *action set, state space,* and *strategy.*

The *action set* of the dialog system includes all possible actions it can perform, such as interactions with the user (e.g., asking the user for input, providing a user some output, confirmations, etc.), interactions with other external resources (e.g., querying a database), and internal processing. For example, the action set in our tutorial dialog system includes the following four actions:

- a question asking for the value of the day (A_d);
- a question asking for the value of the month (A_m);
- A more open-ended question asking for the value of the date (day and month) (A_{dm});
- A final action, closing the dialog and submitting the form (A_f).

When executing actions A_d, A_m, and A_{dm}, the system first asks the corresponding question, and then activates a speech recognition system to obtain the user's answer. Of course actions can be defined at different levels of granularity. For example the actions A_d, A_m, and A_{dm} can be broken into separate lower level actions, including asking the question, activating the

```
Initialization: $s_{t=0} = s_I$

For Each Iteration $t$: {
    if( $s_t \neq s_F$ )   {
        compute current action $a_t$ according to the strategy
        execute $a_t$
        update current state
        $t = t + 1$
    }
    else
        END
}
```

Fig. 1. Description of a sequential decision process.

speech recognizer with the appropriate grammar, getting input from the speech recognizer, and parsing it. In this example, we choose to group these elemental actions into a single higher level action because the order in which they have to be executed is clear and fixed (i.e., first ask, then activate the recognizer, etc.). It is up to the designer to choose the actions and their level of granularity.

The *state* s of a dialog system includes the values of all the relevant internal variables that determine the next action that the system executes. Given the same external conditions (i.e., user responses, database results, etc.) the system future behavior is uniquely determined by the current state. The state space can be finite or infinite, and it includes special initial (s_I) and final (s_F) states. For our simple tutorial example, the state description has to include at least two integer variables that describe the day (d) and the month (m), whose values can be either zero (i.e., the corresponding variable is unassigned), or assigned to the answer provided by the user. For this state representation we have a total of 411 possible states, including one initial state ($d = 0$, $m = 0$), 12 states for which the month variable is assigned and the day is not ($d = 0$, $m = 1, \cdots, 12$), 31 states in which only the day variable is assigned ($d = 1, \cdots, 31$, $m = 0$), 366 states with complete dates, and a special final state ($d = -1$, $m = -1$). Again, it is up to the system designer to decide which variables should be included in the state. For example, in a more sophisticated system, we could have included in the state some information about past values of day and month variables obtained during the same interaction. As we will see in the next section, sometimes it is necessary to expand the state description with additional variables in order to be able to model the system as a Markov decision process.

When an action a is taken in state s, the system state changes. For the day-and-month example, when the system is in an initial state and it asks the user for the month (action A_m) the next state depends on the actual answer of the user as well as on the speech recognition performance, and it can be any one among the 12 states ($d = 0$, $m = 1, \cdots, 12$) in which only the month is filled, assuming that the grammar of the speech recognizer is constrained to recognize only months at this stage of the dialog. We will discuss more about state transitions in the section about Markov decision process (MDP).

A *dialog session* corresponds to a path in the state space starting at the initial state and ending at a final state.

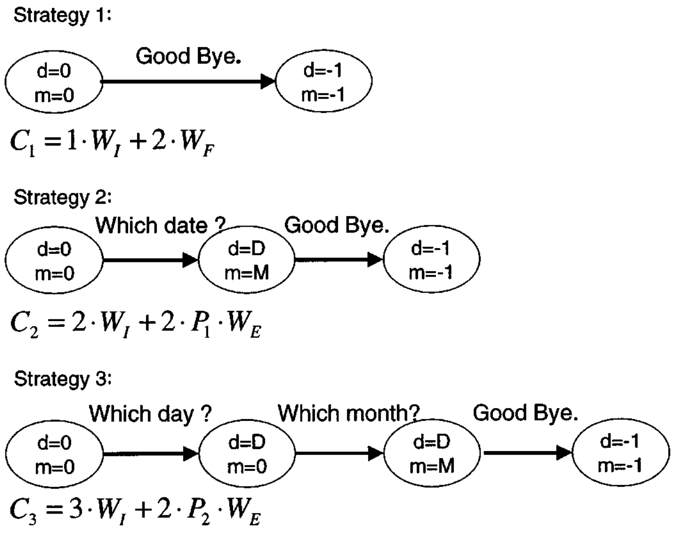

Fig. 2. Three posible strategies for the day-month dialogue system.

A *dialog strategy* specifies, for each state reached, what is the next action to be invoked.

Fig. 1 summarizes the description of a dialog system as a sequential decision process. The dialog system starts a new interaction in the initial state. It chooses the current action from its set of possible actions according to the strategy. As the result of the performed action, the dialog makes a transition to the next state. This process repeats until the final state is reached. Of course, different strategies for the same system result in different values of the objective function (1). Fig. 2 shows three different strategies for the day-and-month system. For strategy 1, where the system immediately ends the dialog, the value of the objective function is $1 \cdot W_i + 2 \cdot W_f$ since there is one interaction and two unassigned variables in the submitted form. For strategy 2, where the system opens with an open-ended question (*Which date?*), sets the values of the day and month variables according to the output of the speech recognizer, and ends the dialog, the expected value of the objective function is $2 \cdot W_i + 2 \cdot p_1 \cdot W_e$. Here we assume that the recognizer has a probability p_1 of making an error in the recognition of the month or day value. For strategy 3, where the system asks separately for a day and a month, the expected value of the objective function is $3 \cdot W_i + 2 \cdot p_2 \cdot W_e$, where p_2 is the error probability of the speech recognizer when specific questions about month and day are asked. Usually, p_2

is smaller than p_1 because the speech recognizer can use a more constrained grammar in the case of a more restricted question.

We define an *optimal strategy* as the one that minimizes the objective function. For example, strategy 1 (where the system does not even engage in dialog, closing the dialog as the first action) is optimal when the recognition error rate is too high, i.e., $p_2 > (W_f - W_i)/W_e$. Strategy 3 is optimal when the difference in error rates justifies a longer interaction, i.e., $p_1 - p_2 > W_i/2W_e$. Note that if we would include other variables in the state, like the value of day and month obtained during past interactions in the same session, we would have to consider also strategies in which the same question can be repeated more than once.

III. Dialogue as a Markov Decision Process

Until now, we have described a dialog system as a sequential decision process in terms of states, action set, and strategy. We showed that with the objective function (1) it is possible to evaluate the performance of each proposed strategy. This is useful when a designer comes up with a few reasonable strategies and is interested to know which one is the best among them. However, we would like to address the more difficult problem of finding the optimal strategy automatically. Since the number of possible strategies is exponential (i.e., $N_a^{N_S}$, where N_a is the number of actions and N_S is the number of states), it is impractical to exhaustively rank all of them. Although the sequential decision process framework introduced for dialog in the previous section is completely general, it does not allow to find the optimal strategy automatically without making some additional assumptions. In this section, we introduce the two assumptions necessary to describe a dialog system as an MDP for which techniques exist for finding the optimal strategy. The first assumption concerns assigning a probabilistic model to state transitions. When an action a_t is taken at time t while in state s_t , the MDP state changes to s_{t+1} according to transition probabilities with the following Markovian property:

$$\begin{aligned} &P(s_{t+1}|s_t, s_{t-1}, \cdots, s_0, a_t, a_{t-1}, \cdots, a_0) \\ &\quad = P_T(s_{t+1}|s_t, a_t). \end{aligned} \tag{3}$$

For the day and month dialog, for instance, when the system is in the initial state, and the action *Which month?* is selected, we model the next state distribution as zero for all but the 12 states in which the month value is given and the day is not. The probability for these states is determined by a prior over the month (i.e., the probability that a specific month is picked by a user) and the confusion matrix of the recognizer (i.e., the probability that a month i is recognized while month j is spoken). The assumption underlying this model is that the user always complies with the system (i.e., always provides, as a response, only the information he/she was asked for), and his answer does not depend on any other information but the one described by the current state and action pair [the Markovian property (3)]. This is a reasonable assumption in our case. In other cases, in order to satisfy the Markovian property (3), the state description has to be padded with extra information (for example, the previously recognized answers of the user to the same question, number of turns in the dialog until the current one, etc.)[2]

The second assumption concerns stochastic modeling costs. When action a_t is executed while in state s_t the system receives a feedback cost c_t distributed according to

$$P(c_t|s_t, s_{t-1}, \cdots, s_0, a_t, a_{t-1}, \cdots, a_0) = P_C(c_t|s_t, a_t). \tag{4}$$

If we define the *session cost* as a sum of all the costs experienced by the system during a dialog session (a path in the state space starting in the initial state, and ending in the final state), the objective function for MDP is the expected session cost.[3] Therefore, in order to describe a dialog system as an MDP, we need to assign cost distributions such that the expected dialog session cost will be the objective function (1), i.e

$$\left\langle \sum_{t=0}^{T_F} c_t \right\rangle = C \tag{5}$$

where T_F is the time step at which the final state s_F is reached, i.e., $s_{T_F} = s_F$. Sometimes, in order to satisfy (4) and (5), the state space description has to be altered. In our example it is impossible to come up with cost assignments that would satisfy both (4) and (5)[4], and the state description has to be extended in order for the Markov property to apply. The reason for that is that the probability of error (and hence the cost) in the value of the assigned variable depends on the question that was used. Besides, if during the dialog several actions were used that resulted in reassigning the same variable, only the last one (the one that assigned the value submitted upon the completion of the dialog) should be taken into account. Since the cost cannot depend on past information except the current state/action pair (4), the state description should have an indication of what action was used to assign the value of each variable. One simple way of doing that is including 2 extra bits of information (q_d and q_m) in the state description, one for each variable, indicating whether the corresponding value was obtained with a simple (e.g., *Which day?* – $q_d = 0$, *Which month?* – $q_m = 0$) or composite (e.g., *Which date?* – $q_d = 1$ and $q_m = 1$) question. If a variable is not assigned, the value of the describing bit is irrelevant. Although this extension quadruples the number of states, the transition probabilities do not change radically since q_d and q_m are assigned deterministically in the next state depending on the type of the current action. The cost assignment in this case can be then described with the following cost distributions.

Any time a question is asked (actions A_d ,A_m, and A_{dm}) the system incurs a constant cost W_i with probability 1:

$$c(A_d) = c(A_m) = c(A_{dm}) = W_i$$

[2]Another related model, called *partially observable MDP* (or POMDP [6]), provides a different way of handling non-Markovian state transitions by modeling the fact that the state of the underlying MDP is not fully observed by the dialog system.

[3]We are describing here the so-called undiscounted MDP.

[4]There is an error in the cost assignment in this example published in [20].

When the dialog is closed (action A_f) the cost is $W_i + N_e W_e + N_f W_f$, where N_e is the number of errors and N_f is the number of unassigned variables in the current state. If we assume that the error rate for the recognition of the month or day values is p_2 for simple questions and p_1 for composite questions, with $p_1 > p_2$, then the cost distribution for closing the dialog is as follows.

- For the initial state, i.e., both variables unassigned: $c(s_I, A_f) = W_i + 2W_f$ with probability 1.
- For all states s with only one variable assigned with a simple question (either $d \neq 0,\ m = 0,\ q_d = 0$ or $d = 0,\ m \neq 0,\ q_m = 0$)

$$c(s, A_f) = \begin{cases} W_i + W_f + W_e, & \text{with prob. } p_2 \\ W_i + W_f, & \text{w.p. } 1 - p_2 \end{cases}. \tag{6}$$

- For all states s with both variables assigned by composite questions (i.e., $q_d = 1$ and $q_m = 1$)

$$c(s, A_f) = \begin{cases} W_i + 2W_e, & \text{w.p. } p_1^2 \\ W_i + W_e, & \text{w.p. } 2p_1(1 - p_1) \\ W_i, & \text{w.p. } (1 - p_1)^2 \end{cases}. \tag{7}$$

- For all states s with both variable assigned by the simple question (i.e., $q_d = 0$ and $q_m = 0$)

$$c(s, A_f) = \begin{cases} W_i + 2W_e, & \text{w.p. } p_2^2 \\ W_i + W_e, & \text{w.p. } 2p_2(1 - p_2) \\ W_i, & \text{w.p. } (1 - p_2)^2 \end{cases}. \tag{8}$$

- For all states s with both variables assigned, one by a simple question and one by a composite question (i.e., $q_d = 0,\ q_m = 1$ or $q_d = 1,\ q_m = 0$)

$$c(s, A_f) = \begin{cases} W_i + 2W_e, & \text{w.p. } p_1 p_2 \\ W_i + W_e, & \text{w.p. } p_1(1 - p_2) + p_2(1 - p_1) \\ W_i, & \text{w.p. } (1 - p_1)(1 - p_2) \end{cases}. \tag{9}$$

This four-tuple composed of state space, action set, transition probabilities, and cost distributions defines an MDP.

IV. Finding the Optimal Strategy

With the assumptions of the previous section, the problem of dialog design is reduced to that of finding the optimal strategy in a MDP that represents the dialog system. MDP's are well known in computer science and machine learning communities, and are used for diverse applications like games [18], telecommunications [28], scheduling [34] and control [22]. Moreover there exists a field known as *reinforcement learning* devoted to research on algorithms aimed at finding the optimal strategy for MDP's. Although we do not attempt here to give a comprehensive review of the field, we will try to give some intuition concerning the related issues. The nature of our application (i.e., dialog system) leads to a special case of MDP called finite horizon, or episodic MDP, where there is an upper bound for the duration of a session (in number of actions). This is based on the assumption that even if the dialog system fails to complete its task in a reasonable time, user's' patience will eventually limit the length of the dialog. For simplicity, we will refer in this section to this case only. For a good review of the field, see [6] and [31].

The first important result is that in MDP an optimal strategy is a *policy,* which is a mapping between states and actions. This means that in MDP, in order to decide which is the optimal action to take in a given state, there is no need to consider other information except that included in the state itself (e.g., there is no need to consider past states and actions).

Due to the Markov properties (3) and (4), several dynamic programming techniques exist for computing the optimal strategy π^* given the correct model parameters. These techniques rely on the following definition.

The *optimal value* $V^*(s)$ of a state s is the expected sum of costs incurred from state s and following the optimal strategy π^* until the final state s_F is reached:

$$V^*(s) = \left\langle \sum_{t=0}^{T_F} c(s_t, a_t) \right\rangle \tag{10}$$

where $s_0 = s,\ s_{T_F} = s_F,\ a_t = \pi^*(s_t),\ s_{t+1}$ is a random variable drawn from $P_T(s_{t+1} | s_t, a_t)$, and the expectation is with respect to both P_T and P_C.

The optimal value function is unique and can be defined as the solution to the simultaneous equations

$$V^*(s_t) = \min_a [\langle c(s_t, a) \rangle + \sum_s P_T(s_{t+1} = s | s_t, a) V^*(s)]. \tag{11}$$

Equation (11) states that the optimal value of state s_t is a sum of expected instantaneous costs plus the expected value of the next state, using the best available action. Given the optimal value function, the optimal strategy can be computed simply as

$$\pi^*(s_t) = \arg\min_a [\langle c(s_t, a) \rangle + \sum_s P_T(s_{t+1} = s | s_t, a) V^*(s)]. \tag{12}$$

Techniques like value iteration, policy iteration, and others (for further references look at [31]) iteratively solve (11) and (12) for computing the optimal strategy.

A. Learning the Optimal Strategy

It is not always possible to use the above methods for finding the optimal strategy due to one or more of the following reasons. Sometimes the state space is very large (or infinite). The number of equations in (11) equals the number of states, and in this case it is not only impossible to solve them, but even to store in memory the optimal strategy. In addition, the MDP parameters (i.e., transition probabilities and cost distributions) may not be known in advance. As illustrated by the tutorial example, some of the model parameters reflect the probability of user's response given the system question and the state of the dialog. Other parameters can reflect the properties of external resources, such as databases, that are also unknown in advance. In this case, we need to resort to learning.

It is impossible to use supervised learning with a given corpus of interactions to estimate the parameters or the optimal strategy of a dialog system. In fact, supervised learning techniques are based on the assumption that both the training and the test examples are drawn independently from the same distribution. In the case of MDP this assumption is incorrect if the training data is composed of complete sessions produced using a fixed strategy, like in the case of dialog, since the strategy itself determines the distribution of states in the corpus. Therefore, even though we could in principle use supervised learning to estimate the strategy in the corpus very accurately, a small deviation in the learned strategy might produce a distribution over the state space completely different from the one observed in the training set, for which we do not have accurate prediction of strategy. In reinforcement learning, the optimal strategy (or the model parameters) is learned not from a static corpus but through interaction.

The major features of reinforcement learning algorithms are as follows.

- *Exploration:* Since learning is performed while interacting, the exploration of the state-action space can be dynamically controlled by the learning algorithm.
- *Delayed reinforcement:* since the costs incurred in any stage of the dialog are propagated back to all preceding state-action pairs, reinforcement learning can deal with delayed feedback. This is especially important in dialog applications since in many cases the success of a dialog can be measured only at the end of the transaction.
- *Adaptation:* Since the system learns from interaction, it can keep adapting for slowly changing external conditions.

The main problem in using reinforcement learning for human-machine dialog applications is that it requires a large number of dialog sessions for a successful learning. Moreover, for proper exploration of the state-action space the system should sometimes take actions that may not be reasonable for the current situation, making it a cruel and expensive procedure for the users experimenting with the system. To overcome this limitation, we propose the use of a simulated user. The simulated user is a stochastic generative model that produces speech acts as a response to a dialog action. The parameters of the simulated user should be estimated from an annotated dialog corpus. A detailed example of a simulated user for the air travel information system (ATIS) task will be described in the next section. Once a simulated user is available, it can be used in a generative mode for interacting with the dialog system while the reinforcement learning algorithm is estimating the optimal strategy. Then when a reasonable estimate of the optimal strategy is obtained, the system can be used with real users and the learning process can continue. The simulated user is also very valuable for extensive and inexpensive testing of a dialog system looking for bugs and strategy errors [8].

Fig. 3 summarizes the suggested learning paradigm. The dialog corpus is used for estimating the parameters of a stochastic simulated user. The simulated user is used in generative mode for running as many dialog sessions as needed for the reinforcement learning algorithm to estimate the optimal strategy.

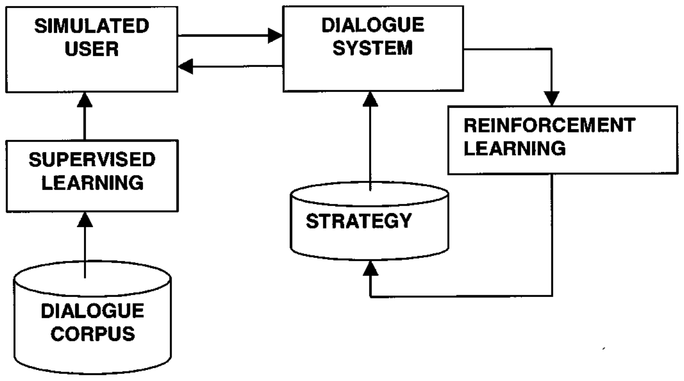

Fig. 3. Procedural description of the learning paradigm.

B. Reinforcement Learning

The reinforcement learning discipline includes many algorithms for finding the optimal strategy. In this paper, we will describe only the one used in the reported experiment, namely Monte Carlo with exploring starts. For a review of other algorithms, see [6] and [31]. The goal of the algorithm is to estimate the optimal state-action value function $Q^*(s_t, a_t)$ defined as the expected cost of a session starting in state s_t, taking action a_t and thereafter proceeding according to the optimal strategy until a final state is reached. Therefore

$$Q^*(s_t, a_t) = \langle c(s_t, a_t)\rangle + \sum_s P_T(s_{t+1} = s | s_t, a_t) V^*(s), \quad (13)$$

$$V^*(s) = \max_a Q^*(s, a) \quad (14)$$

and

$$\pi^* = \arg\max_a Q^*(s, a). \quad (15)$$

This algorithm (like many other reinforcement algorithms) is an iterative procedure beginning with an initial guess of $Q^*(s, a)$ and successively improving it at each iteration. Denoting by $Q_k^*(s, a)$ the estimate at iteration step k, the successive improved estimation is obtained with the algorithm described in Fig. 4.

V. Reinforcement Learning in the ATIS Task

The ATIS dialog task is based on the DARPA ATIS project [1], [10]. It consists of a spoken language interface to an airline database that includes information such as flight schedules, fare, ground transportation, etc. The original ATIS task involves building a user initiated dialog system where the user specifies a database query in spoken natural language and the system displays the data corresponding to the interpreted query. The strategy of a user initiated system is very simple: get the user's input, interpret it in the context of the past inputs, build a database query according to that interpretation, and finally present the result of the query. Although this simple strategy was easy to evaluate objectively (i.e., comparing each retrieved data with reference *correct* data), in many cases the system was not efficient. For instance when a user asked, "I'd like to have information about flights," the correct response was to retrieve all the

```
Initialization:
    For all $s$ and $a$
    $Q_0^*(s,a)$ = arbitrary,
    $\pi_0^*(s) = \arg\max Q_0^*(s,a)$ ,
    $n(s,a) = 0$.
For each iteration k:
    For every s and a:
        generate a session starting at $s$ with action $a$ and proceeding
        according to $\pi_{k-1}^*(s)$ until a final state is reached.

        For each pair $s'$, $a'$ in the session compute the sum of the costs
        $C(s',a')$ following $s$ until the final state and update $n(s',a') = n(s',a') + 1$,
        $Q_k^*(s',a') = C(s',a') \cdot \frac{1}{n(s',a')} + Q_{k-1}^*(s',a') \cdot \frac{n(s',a')-1}{n(s',a')}$ .

        Update the strategy $\pi_k^*(s) = \arg\max_a Q_k^*(s,a)$ .
```

Fig. 4. Description of the learning algorithm used for the ATIS task.

flights from the database and to present them to the (exasperated) user. On the other hand, if the user's query was over-constrained (e.g., "Show me the flights from Newark to San Francisco with American at 9 in the morning"), the system would just present an empty data set that would be of no help to the user, rather than proposing a perhaps acceptable alternate solution (e.g., there is no flight at 9:00 with American, but there is a flight at 9:30 with Delta).

In addition to the common task of building a user-initiated system for the ATIS project, several research groups built sophisticated mixed initiative dialog systems [2], [3], [5], [12], [25] that incorporated heuristics and common sense in order to overcome the limitations of a user initiated system, as illustrated by the previous examples.

The goal of the experiment described in this paper is to show that a sophisticated strategy (like one designed by hand [2], [3], [5], [12], [25]) can be learned automatically with a simple objective function, state and action representation.

In the following experiment, we do not deal with the language understanding component; we consider the input to the dialog model to be a semantic representation of the user utterances. For simplicity, we assume that there are no errors in the semantic transcription (this assumption can be relaxed by estimating error probabilities and confusion matrices like in the day and month example).

A. Objective Function

The goal of the flight information system is to provide the user with information about the flights in an efficient way. The efficiency here involves not only the duration of the dialog (in turns) but also the cost of external resources (database retrieval) and the effectiveness of the system output to the user. We will measure those by the following terms of the objective function:

$$C = W_i\langle N_i\rangle + W_r\langle N_r\rangle + W_o\langle f_o(N_o)\rangle + W_s\langle F_s\rangle \qquad (16)$$

where $\langle N_i\rangle$ is the expected length of the whole interaction in number of turns; $\langle N_r\rangle$, the expected number of tuples retrieved from the database during the session, which reflects the cost of retrieving information; and $f_o(N_o)$ is the data presentation cost function with N_o being the number of records that are presented to the user. Generally, $f_o(N_o)$ is zero for N_o smaller than a reasonable value N^*, and increases rapidly thereafter, where N^* depends on the medium used to output information to the user (it is generally small for voice based communication, and higher for display). Finally, F_s is an overall task success measure that we set to be zero, if some data was ever presented to the user during the session, and one, otherwise. Here, we assumed that the data presented to the user matches the user request, i.e. there are not recognition or understanding errors. In a real system such errors should be taken into account by F_s being zero only if *correct* data is presented, or including an additional term that takes into account wrong outputs like in the tutorial day/month example.

B. Actions

The choice of the level of granularity of the actions for dialog systems is left to the designer. On one hand, we could use very elemental actions, like the execution of primitive computational statements (e.g., add, shift, etc.). On the other extreme the whole execution of a subdialog with its fixed strategy can be considered as a single action. The guidelines we followed in choosing the set of actions were dictated by the following consideration: a dialog action has to have an impact outside the system, which includes all the interactions with the resources outside the system (e.g., user, databases, etc.). Moreover, if several actions are known to be executed always in a predetermined sequence, there is no need to consider them separately, but they constitute a single action. In our case, all the actions involving user interaction consists of the following sequence: playing a prompt possibly switching the grammar in the speech recognizer, collecting speech, getting the recognized string from the recognizer, employing the understanding system, and finally getting its output. This sequence is fixed and it is considered to be a single action parameterized by the prompt.

For the ATIS task we considered the following actions.

- Interactions with the user:
  Greeting: it is an open ended question like *How may I help you?*
  Constraining: this includes a set of actions each requesting a particular attribute from the user, as follows
    Constrain airline: e.g., Please specify your favorite airline.
    Constrain origin: e.g., Where are you leaving from?
    Constrain destination: e.g., Where do you want to go?
    Constrain departure time: e.g., When do you want to leave?
  Relaxing: this includes a set of actions each requesting the user to relax a particular constraint that was specified earlier.
    Relax airline: e.g., Do you mind considering other airlines?
    Relax origin: e.g., Do you mind leaving from somewhere else?
    Relax destination: e.g. Do you mind going somewhere else?
    Relax departure time: e.g., Do you mind leaving at a different time?
  Output: e.g., Here are the 25 flights that match your request $\cdots$.
  Closing: closing the dialog, e.g., *Thanks, good bye.*
- Interaction with the database:
  Retrieval: forming a database query from the information included in the current state and getting the corresponding data.

C. State Representation

The choice in defining a state representation for a given application is determined by the following considerations. The state space should be kept as small as possible. Most of the tabular reinforcement learning algorithms converge in linear time with the number of states N in the underlying MDP [31] . Recently there have been successful applications with extremely large or infinite state space [18], but theoretical bounds on convergence in such cases are not available yet. On the other hand the state representation should contain enough information so that the underlying process is Markovian as specified by (2) and (3).

The tradeoff implied by these considerations led to the following representation for the state of the ATIS dialog system.

The state included three *templates* (a template is a set of keyword-value pairs, called tokens, that was used in our ATIS understanding system [17] as the semantic representation). The **user** template represents the meaning of the user request interpreted in context; the **data** template describes the data retrieved from the database according to the query based on the user template; and the **system** template represents some history of system actions during the interaction. The possible keywords in the user template correspond to the attributes of the database. Since we restrict the task to flight information only, the relevant attributes include:

- *ORIGIN* of the flight;
- *DESTINATION* of the flight;
- *AIRLINE*;
- *DEPARTURE TIME;*
- *ARRIVAL TIME.*

A typical user template looks like:

ORIGIN: EWR
DESTINATION: SFO
AIRLINE: DL

corresponding to the user query, *Show me the flights from Newark to San Francisco with Delta Airlines.* The user template is updated each time an action involving interaction with the user is performed. In addition the user template is used to build the database query when the retrieval action is taken.

In order to keep the number of states finite and small, we further simplified the state representation by removing the value information of each token, therefore taking into consideration only the attributes that are present in the query and not their actual value. This in effect groups the states into a finite number of classes of states with the same value function. The values of the attributes are preserved only for the purpose of database query.

A data template represents the data retrieved from the database. A typical data template looks like:

NDATA: 3
DATA: flight: DL102 *departure time:* 1000 *arrival time:* 1800
DATA: flight: DL68 *departure time:* 1035 *arrival time:* 1903
DATA: flight: DL99 *departure time:* 1305 *arrival time:* 2015

and includes a token representing the number of data tuples retrieved (NDATA) and one token for each tuple. Again for limiting the number of states, we ignore the actual retrieved tuples (they are used only for user's output), and keep only the NDATA token. Moreover, the value of NDATA is quantized into a finite number of intervals (LOW–MEDIUM–HIGH–VERY HIGH).

The system template is needed for recording a partial history of actions in order to maintain the Markovian properties. In our case it will contain the keyword OUTPUT only if an output action in a state with nonzero tuples (NDATA $>$0) was taken in the course of the current interaction.

D. Costs

The cost distributions that correspond to the objective function (16) are as follows.

- $c(s, a) = W_i$ for all s, and when a is an action involving interaction with the user, except the data output and closing actions (i.e. constraining questions, greeting, relaxation).
- $c(s, a) = W_r N_r$ for all s and when a is the retrieval action with N_r the number of tuples retrieved. Since the state representation does not contain the actual value of the user specified constraints, N_r is not a deterministic function of s. For example two user requests such as *flights from Boston to San Francisco with United* and *flights from Denver to Atlanta with Delta* map to the same state since the values of the attributes ORIGIN, DESTINATION, and

AIRLINE are omitted in the state representation. In effect we trade off a larger state space for nondeterministic costs. This nondeterminism in costs can be represented by a distribution that depends solely on the database statistics and the prior on user choices for the value of constraints. In principle, if the state representation would include values of constraints as well (therefore resulting in a very large or infinite state space), the optimal dialog strategy would depend upon specific users' choices. For instance, the system could learn that if the user asks about flights *from New York to San Francisco,* more information needs to be requested prior to database access for reducing the database access and presentation costs; that information will not be necessary, for instance, for flights *between Cape Cod and Boston.*

- $c(s,a) = W_o f_o(N_o)$ for all s and when a is the data output action, where N_o is the number of tuples being presented to the user and is represented in the state s by the token with keyword NDATA. If NDATA is not present in s, the value of N_o is 0. The function f_o is defined as
$$f_o(n) = \begin{cases} 0, & n < N^* \\ w \cdot (n - N^*), & n \geq N^* \end{cases}. \tag{17}$$
- $c(s,a) = W_s F_s(s) + W_i$ for all states s and action a being the closing action. $F_s(s)$ is zero for all states that include the keyword OUTPUT in the system template, and one, otherwise.

E. State Transitions and the Simulated User

The state transitions that involve user interaction are not deterministic, since they depend on the actual input of the user. The only deterministic transition is the one out of the final action (closing) that always leads to the final state. The set of probabilities that describe the state transitions that result from the user response is called the *simulated user.* For this experiment, the simulated user is a generative stochastic model that given the system's current state and the current actions (i.e., the prompt) produces the semantic representation of an utterance in a template form similarly to the user's template described in Section V-C. This template is then merged with the user template of the current state to produce the user template of the next state. The merge follows two simple rules:

- new values of the same keyword override old ones, otherwise they are added;
- if in the current interaction the user agrees to relax an attribute, that attribute is deleted from the merged template.

We chose the model of the simulated user in such a way that the parameters could be easily and reliably estimated from a corpus of dialogs. This involved the assumption that the user response depends only on the current system action (i.e., the prompt) and not the state. The only dependency on the current state results from the following consistency assumption: if the current user response includes a repetition of previously specified attribute (hence already included in the current state), its value is consistent with the one in the state. Although it would be possible to model the user's *change of mind,* we did not include that in the current implementation.

We parameterized the simulated user in the following way.

1) Response to Greeting: Includes the probability $P(n)$, $n = 0, 1, 2, \cdots,$ where n is the number of attributes specified by the user in a single utterance corresponding to the number of tokens in the user template; the probability distribution $P(attribute)$ (e.g., ORIGIN, DESTINATION, AIRLINE, $\cdots$), and the probability distribution on the value of each attribute [e.g., *P(Boston|ORIGIN), P(Delta|AIRLINE)*] We assume that the attributes are chosen independently while each new attribute and its value are merged with the previous ones according the merge rules.

2) Response to Cnstraining Questions: Parameterized by $P(k_R|k_G)$, i.e., the probability of the user specifying a value for attribute k_R when asked for the value of attribute k_G. For instance *P(airline|departure time*) is the probability of the user specifying a preferred airline when asked for desired departure time. Additional parameters are $P(N|k_G)$, i.e., the probability of providing N additional unsolicited attributes in the same response. We assume that he additional attributes are generated independently using the same distribution as in the response to greeting.

3) Response to a Relaxation Prompt: Parameterized by $P(yes|k_G) = 1 - P(no|k_G)$, i.e., the probability of accepting (or rejecting) the proposed relaxation of attribute k_G.

As mentioned before, in this experiment we assume that recognition and understanding make no errors. For a real system, such errors can be simulated with parameters representing error probabilities and confusion matrices.

The only available data we had on flight domain was the ATIS original user initiated dialog corpus [9], and therefore we could estimate only the parameters that characterize the response to greeting. Since in this corpus the system never takes initiative, and does not ask constraining or relaxing questions, we set the other parameters arbitrarily. For example, the $P(k_R|k_G)$ matrix had a high valued diagonal (indication that the user is likely to comply with the syste request), but nonzero off-diagonal terms, especially for attributes that are likely to be of equal importance (like departure time versus arrival time). In addition we assumed that the user is less likely to agree to relax primary attributes such as *origin* and *destination,* than secondary ones like *airline* and *departure time.*

In principle, all these parameters could have been estimated using a mixed initiative dialog corpus. The estimation of user parameter does not need to be precise. Recent theoretical results [27] show that a very low accuracy in the estimation of transition probabilities is required in order to find a good approximation to the optimal strategy.

F. Experimental Results

The goal of our experiment is to show that a nontrivial strategy can be automatically learned given the simple objective function (16). The system was initialized with no knowledge of the task, i.e., each action had the same probability of being selected by the system at any step. A typical dialog with an untrained system will look like the one in Fig. 5 (in this and in the following example the English responses of the user and the system are included only for illustration purpose—the

S: RELEASE AIRLINE	*Do you want to choose another airline?*
U:	What?
S: RETRIEVAL	
S: CONSTRAIN DEPART. TIME	*When do you want to leave?*
U:	UH? I want to go to Boston
S: OUTPUT DATA	*There are 23,457 flights...*
U:	I don't understand, can you please show me the flights from San Francisco to Boston?
S: CLOSE DIALOGUE	*Thank you for using AT&T.*

Fig. 5. Example of dialogue with an untrained system (S = system, U = user).

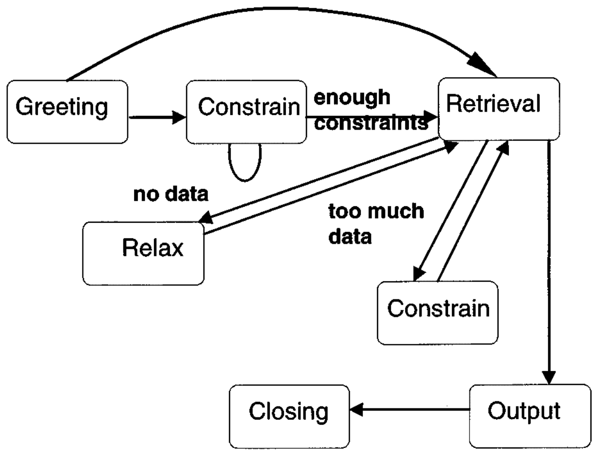

Fig. 6. Schematic representation of the optimal strategy for ATIS.

user and the system communicate through semantic templates). Of course, the total cost for this dialog is quite high, especially due to the high cost of retrieval (all the database was retrieved here) and data output (23 457 flights).

We used a Monte Carlo training algorithm with exploring starts described in Fig. 4 and ran it for 100 epochs (one epoch includes dialogs starting in all possible states explored so far with all possible actions). By the end of the training the system explored 111 states[5], and converged to the optimal strategy schematically described by Fig. 6. In the optimal strategy the system always starts the dialog by *greeting.* Depending on the system state after getting the user response to greeting, the system, if needed, proceeds by asking *constraining* questions until the origin, destination, and airline are specified. Then it *retrieves* data from the database. After the retrieval, if the resulting data set is empty (because the query was over-constrained) the system, depending on the current state, *relaxes* the airline or the departure time, and then *retrieves* again. If there are too many flights in the data set, it asks for additional *constraints* (e.g., the departure time) and then *retrieves* again. If at any point during the dialog the retrieved data set has a reasonable number of flights, the data is *output* and the dialog

[5]As explained in Section V-C, a state in this experiment represents the cluster of all the original states that have the same keywords in the user template and the same range of the number of tuples in the data template.

S: GREETING	*This is the AT&T flight information. How may I help you?*
U:	I want to go to Boston.
S: CONSTRAIN ORIGIN	*Where do you want to leave from?*
U:	San Francisco
S: CONSTRAIN AIRLINE	*Which airline?*
U:	Delta
S: RETRIEVAL (30 flights)	
S: CONSTRAIN DEPART. TIME	*When do you want to leave?*
U:	In the late afternoon.
S: RETRIEVAL (3 flights)	
S: OUTPUT DATA	*Flight ... leaves at ..., flight ... leaves at ...*
U:	Thanks.
S: CLOSE DIALOGUE	*Thank you for using AT&T.*

Fig. 7. Example of dialogue with a trained system.

is *closed.* An example of a dialog performed with a trained system is shown in Fig. 7.

The strategy we just described reflects the objective function in the following way. The system learned to start every dialog with a greeting because by doing so it maximizes the number of constraints provided with a single exchange (the simulated user has a higher probability of giving more constraints after greeting than after any other prompt), thus minimizing the duration term. The constraining behavior after the greeting is induced by the minimization of the retrieval cost: the system does not query the database until enough constraints are gathered. The constraining behavior after the retrieval minimizes the data presentation cost by not allowing the output of too many tuples. The relaxation results from the minimization of the task success cost by forcing the system to output data. Since the different costs are interdependent, it is incorrect to minimize them independently, and the optimal strategy trades them off in order to optimize the combined objective function (16). For example, minimizing only the cost of retrieval would result in dialogs where no retrieval is done and no information is provided to the user as a result.

Fig. 8 shows the value of the objective function (16) as a function the epoch number. While learning and before reaching the optimal strategy described above, the system went through the four strategies schematically shown in Fig. 9.

Strategy 1 was learned after only a few dialogs during the first epoch. In this strategy, the system immediately closes the dialog, and its cost is quite high due to the user dissatisfaction cost, but it is much lower than the cost of a random strategy as shown in the first dialog above. The second strategy that the system learns corresponds to the user initiated strategy, as in the original ATIS system: it opens the dialog by greeting the user, retrieves data from a database according to the user's request, outputs the data to the user, and closes the dialog. In the third strategy, the system learned that if too much data is retrieved,

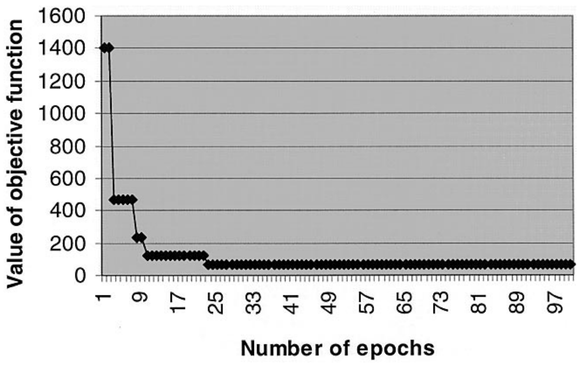

Fig. 8. Convergence of the learning algorithm for the ATIS dialogue.

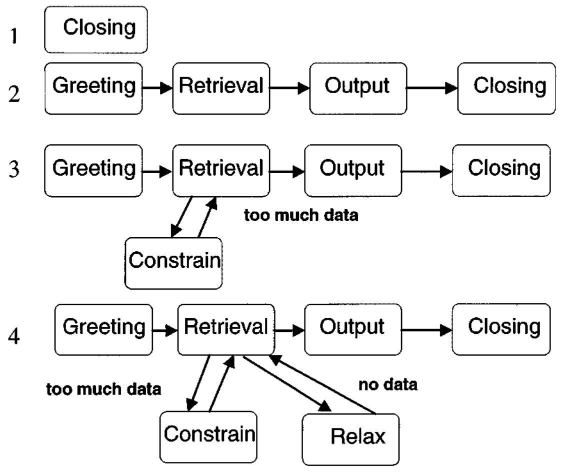

Fig. 9. Schematic representation of incrementally more complex strategies obtained during the training.

then it should ask the user constraining questions about airline and departure time. It takes ten epochs (about 1000 dialogs) to learn strategy 4. Here the system learns to relax constrains (departure time, airline) if the retrieval resulted in an empty data set. The optimal strategy of Fig. 5 is learned after 23 epochs. The last thing the system learns is to gather enough information from the user *before* the data retrieval. The rate of convergence (i.e., the number of dialogs needed for the system to learn) and the actual sequence of strategies the system goes through depend on the flavor and the parameters of the learning algorithm used.

G. Summary and Discussion

The main thesis of this work is the formalization of dialog as an optimization problem. The objective function is a weighted sum of costs representing different dimensions of the dialog quality: distance from task achievement, efficiency in terms of dialog turns, quantity of information exchanged, cost of external resources, effectiveness of presentation, etc. Even costs related to abstract measures such as user satisfaction can be modeled in the same form [9]. The result of the ATIS experiment (Section V-F) shows that a complex strategy similar to one that was developed independently under heuristic considerations by different groups [2], [3], [5], [12], [25] results from the optimization of a relatively simple objective function.[6] We believe that we can model the *common sense* that dialog designers are using while building a system by an explicit criterion. The bulk of the dialog design process involves performing implicit optimization by writing common sense rules. Instead we propose that the dialog design process should involve the design of the criterion, while the optimization can be done automatically (computers are generally better optimizer than humans and are able to find better solutions).

We also show that any dialog system can be formally described as a sequential decision process in terms of its state space, action set, and strategy. With additional assumptions about the state transition probabilities and cost assignment, a dialog system can be mapped to a stochastic model known as Markov decision process (MDP). A variety of algorithms for finding the optimal strategy (i.e., the one that optimizes the criterion) is available within the MDP framework. We are interested in data-driven algorithms that learn the optimal strategy from examples. The known problem of using supervised learning from a corpus of dialogs results from the dynamic nature of dialog itself. Once the system deviates from the dialog in the corpus, there is no way of evaluating and finding the correct supervision for the rest of the transaction. Learning is not supervised but rather by reinforcement in the MDP framework: the system actively interacts with users, exploring different portions of the state space, and learns by attributing values to the different actions/state pairs according the final cost of each dialog. However, the exploration of the state space can be rather costly, involving many dialogs until the system learns a reasonable, or close to optimal, behavior. Moreover, in the exploration phase of learning, some of the actions the system tries might make no sense to normal users. To overcome these problems we propose to use a simulated user: a stochastic generative model parameterized in such a way that it can be reliably estimated using supervised learning on a dialog corpus. Such a simulated user is useful also for debugging a dialog system in the early stages of design and finding bugs in the strategy. The actual parameterization of the user model should influence the way we collect corpora of dialogs.

The experimental results we present in this paper show that it is indeed possible to find a simple criterion, a state space representation, and a simulated user parameterization in order to automatically learn a relatively complex dialog behavior, similar to one that was designed by intuition.

The important open questions in the dialog learning and evaluation within the MDP paradigm are as follows.

- A principled way of finding a good objective function for a given task. In many applications the dimensions that measure dialog performance are clear, but the weights specifying the desired tradeoff among them is not known. In principle, for a commercial application, the weighted cost terms should equal their value measured in currency (e.g., U.S. dollars). As discussed earlier, some of the cost terms represent abstract and not directly measurable dimensions, such as user satisfaction. These abstract dimensions can be in turn represented as a linear combination of directly measurable quantities as in [9]. Although it was shown in

[6]This is a qualitative statement. A quantitative comparison is not possible since the handcrafted systems were not based on an objective function for evaluation.

[9] that is possible to learn the weights of the terms representing the user satisfaction through the user's feedback, often the user satisfaction will represent only one of the terms of the criterion (1), and an open question remains on what is a principled way of estimating the weights.

- In the examples shown in this paper, the dialog state space was carefully handcrafted in order to satisfy the Markov properties (2) and (3) required within the MDP paradigm. The chosen state representation imposes certain restrictions on the structure of the simulated user. In fact, due to the Markov property, the simulated user's output is independent of anything except current state and current dialog action. Since the resulting optimal strategy reflects simulated user structure and parameters, the choice of state representation plays a crucial role in dialog strategy learning. In order to automatize dialog design, it is important to look at data-driven techniques for state estimation.
- The learning algorithm used here required a large number of interactions; a more efficient learning is desirable. Starting from *tabula rasa,* as we did in the described experiment, is often not necessary. Knowledge about the task can be incorporated in the initial strategy, limiting significantly the necessary exploration of the state space. There is a bulk of research in the machine learning community dealing with the issue of more efficient learning, using macro actions [29] (representing subgoals) to speed up learning, etc. Many of the issues under investigation apply naturally to the specific problem of dialog design.
- The stochastic paradigm is common in automatic speech recognition, and it is gaining popularity in the language understanding community. With the introduction of a stochastic model for dialog, an open question remains of how to integrate these different levels of knowledge and learn the corresponding models in an integrated way.

Acknowledgment

The authors wish to thank S. S. Baveja, M. Kearns, and R. Sutton for their helpful comments on this paper.

References

[1] *Proc. 1995 ARPA Spoken Language Systems Technology Workshop*, Austin, TX, Jan. 1995.
[2] J. Glass *et al.*, "The MIT Atis System: December 1994 Progress Report," in *Proc. 1995 ARPA Spoken Language Systems Technology Workshop*, Austin, TX, Jan. 1995.
[3] E. Levin, R. Pieraccini, and A. Di Carlo, "User initiated and mixed initiative dialog for database information retrieval," in Automated Spoken Dialog Systems, S. LuperFoy, Ed: MIT Press, to be published.
[4] M. D. Sadek *et al.*, "Cooperative spoken dialog system based on a rational agent model: A first implementation on the AGS application," in *Proc. ESCA/ETR Workshop on Spoken Dialog Systems*, Hanstholm, Denmark, 1995.
[5] D. Stallard, "The BBN ATIS4 Dialog System," in *Proc. 1995 ARPA Spoken Language Systems Technology Workshop*, Austin, TX, Jan. 1995.
[6] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, no. 4, pp. 237–285, May 1996.
[7] S. M. Marcus *et al.*, "Prompt constrained natural language—Evolving the next generation of telephony services," in *Proc. ICSLP '96*, Philadephia, PA, Oct. 1996.
[8] W. Eckert, E. Levin, and R. Pieraccini, "User modeling for spoken dialog systems," in *Proc. IEEE ASR Workshop*, Santa Barbara, CA, 1997.
[9] M. A. Walker, D. J. Littman, C. A. Kamm, and A. Abella, "PARADISE: A framework for evaluation of spoken dialog agents," in *Proc. 35th Ann. Meeting Assoc. Computational Linguistics*, Madrid, Spain, 1997.
[10] , "MADCOW: Multi-Site Data Collection for a Spoken Language Corpus," in *Proc. 5th DARPA Workshop on Speech and Natural Language*, Harriman, NY, Feb. 1992.
[11] R. Pieraccini and E. Levin, "Stochastic representation of semantic structure for speech understanding," *Speech Commun.*, vol. 11, pp. 283–288, 1992.
[12] W. Ward, "The CMU air travel information service: Understanding spontaneous speech," in *Proc. 3rd DARPA Workshop on Speech and Natural Language*, Hidden Valley, PA, June 1990, pp. 127–129.
[13] W. Ward and S. Issar, "The CMU ATIS System," in *Proc. 1995 ARPA Spoken Language Systems Technology Workshop*, Austin, TX, Jan. 1995.
[14] R. Kuhn and R. De Mori, "The application of semantic classification trees to natural language understanding," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, pp. 449–460, May 1995.
[15] S. Miller, R. Bobrow, R Schwartz, and R. Ingria, "Statistical language processing using hidden understanding models," in *Proc. of 1994 ARPA Spoken Language Systems, Technology Workshop*, Princeton, NJ, Mar. 1994.
[16] F. Jelinek, "Continuous speech recognition by statictical methods," *Proc. IEEE*, vol. 64, pp. 532–556, 1976.
[17] E. Levin and R. Pieraccini, "CHRONUS, The next generation," in *Proc. 1995 ARPA Spoken Language Systems, Technology Workshop*, Austin, TX, Jan. 1995.
[18] G. Tesauro, "TD Gammon, a self-teaching backgammon program, achieves master-level play," *Neural Comput.*, vol. 6, no. 2, pp. 215–219, 1994.
[19] R. W. Smith, D. R. Hipp, and A. W. Biernmann, "An architecture for voice dialog systems based on prolog-style theorem proving," *J. Comput Linguist.* , vol. 21, pp. 281–320, 1995.
[20] E. Levin, R. Pieraccini, and W. Eckert, "Using Markov decision process for learning dialog strategies," in *Proc. ICASSP 98*, vol. 1, Seattle, WA, May 1998, pp. 201–204.
[21] H. Alshawi, "Head automata and bilingual tiling: Translation with minimal representations," in *Proc. 34th Ann. Meeting Assoc. Computational Linguistics*, Santa Cruz, CA, 1996, pp. 167–176.
[22] E. H. Crites and E. G. Barto, "Improving elevator performance using reinforcement learning," in *Proc. Conf. Neural Information Processing Systems 8*, D. Touretzky, M. Mozer, and M. Hasselmo, Eds., 1996.
[23] V. Zue *et al.*, "From interface to content: Translingual access and delivery of on-line information," in *Proc. EUROSPEECH'97*, vol. 4, Rhodes, Greece, Sept. 1997, pp. 2227–2230.
[24] A. L. Gorin, G. Riccardi, and J. H. Wright, "How may I help you?," *Speech Commun.*.
[25] L. Lamel *et al.*, "The LIMSI RailTel system: Field trials of a telephone service for rail travel information," *Speech Commun.*, vol. 23, pp. 67–82, Oct. 1997.
[26] E. Charniak, *Statistical Language Learning*. Cambridge, MA: MIT Press, 1993.
[27] M. Kearns and S. Singh, "Finite-sample convergence rates for Q-learning and indirect algorithms," in *Proc. Neural Information Processing Systems, Natural and Synthetic*, Denver, CO, Nov. 1998.
[28] S. Singh and D. Bertsekas, "Reinforcement learning for dynamic channel allocation in cellular telephone systems," in *Proc. Neural Information Processing Systems, Natural and Synthetic*, Denver, CO, Nov. 1996, pp. 974–980.
[29] R. S. Sutton, S Singh, D. Precup, and B. Ravindran, "Improved switching among temporally abstract actions," in *Proc. Neural Information Processing Systems, Natural and Synthetic*, Denver, CO, Nov. 1998.
[30] L. R. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*. Englewood Cliffs, NJ: Prentice-Hall, 1993.
[31] R. S. Sutton and A. G. Barto, *Reinforcement Learning An Introduction*. Cambridge, MA: MIT Press, 1998.
[32] E. Barnard *et al.*, "A consistent approach to dssigning spoken-dialog systems," in *Proc. 1999 IEEE ASRU Workshop*, Keystone, CO, Dec. 1999.
[33] J. F. Allen *et al.*, "The TRAINS project: A case study in defining a conversational planning agent," *J. Exper. Theoret. Artif. Intell.*, vol. 7, pp. 7–48, 1995.
[34] W. Zhang and T. G. Dietterich, "High Performance Job-Shop Scheduling with a Time-Delay TD(λ) network," in *Proc. 1995 Conf. Advances in Neural Information Processing Systems*, Cambridge, MA, 1995, pp. 1024–1030.

[35] M. Denecke and A. Waibel, “Dialog strategies guiding users to their communicative goals,” in *Proc. EUROSPEECH’97*, vol. 4, Rhodes, Greece, Sept. 1997, pp. 2227–2230.

[36] A. Abella, M. K. Brown, and B. Buntschuh, “Development principles for dialog-based interfaces,” in *Proc. Eur. Conf. Artificial Intelligence*, Budapest, Hungary, 1996.

Esther Levin received the Ph.D. in electrical engineering from the Technion, Israel Institute of Technology, Haifa, in 1988.

Since 1988, she she has been with AT&T Labs—Research, Florham Park, NJ, and its previous incarnations. Her research interests include data-driven learning methods for natural language understanding and dialog design.

Roberto Pieraccini (M’90) received the Dr.Ing. degree in electrical engineering from the Universita degli Studi di Pisa, Pisa, Italy in 1980.

From 1981 to 1990, he was with CSLET, Torino, Italy, where he worked on algorithms for speech. In June 1990, he joined the Speech Research Group, AT&T Bell Laboratories, and was involved in stochastic methods for language understanding. In 1995 he joined AT&T Laboratories, Florham Park, NJ, where he was Principal Member of Technical Staff. He is currently with SpeechWorks International, Inc., New York, NY. His current interests are in natural language and dialog.

Wieland Eckert (M’97) received the M.S. degree in computer science (Dipl.-Ing.) in 1991, and the Ph.D. degree in engineering sciences (Dr.-Ing.) in 1996, both from the University of Erlangen, Nürnberg, Germany.

Until 1991, he was a Consultant in software development for several companies. From 1991 to 1996, he was a Researcher and Teaching Assistant with the Department of Pattern Recognition, University of Erlangen, where he participated in the EU-founded ESPRIT project Sundial. In this project, he played an important role in developing the first multilingual and multifunctional dialog manager in Europe. In 1996, he joined AT&T Laboratories—Research, Florham Park, NJ, where he worked in the fields of speech dialog systems and user modeling. Since 1999, he has been with Lucent Technologies, Nürnberg, working in the field of local access networks. His research interests are in the fields of human computer interaction and speech dialog systems.