CHAPTER 3

DIALOGUE, REPRESENTATION, INFERENCE AND LEARNING STRATEGIES
In the last many years researchers have designed spoken dialogue system which have the capability to communicate with the users in the real time. Many Spoken Dialogue systems are realized which are the good examples of real time, goal-oriented interactions between humans and computers that perform tasks like finding a good restaurant nearby, reading your email, perusing the classified advertisements about cars for sale, or making travel arrangements (Seneff, Zue, Polifroni, Pao, Hetherington, Goddeau, & Glass, 1995; Baggia, Castagneri, & Danieli, 1998; Sanderman, Sturm, den Os, Boves, & Cremers, 1998; Walker, Fromer, & Narayanan, 1998). Yet in spite of 40 years of research on algorithms for dialogue management in task-oriented dialogue systems, (Carbonell, 1971; Winograd, 1972; Simmons & Slocum, 1975; Bruce, 1975; Power, 1974; Walker, 1978; Allen, 1979; Cohen, 1978; Pollack, Hirschberg, & Webber, 1982; Grosz, 1983; Woods, 1984; Finin, Joshi, & Webber, 1986; Carberry, 1989; Moore & Paris, 1989; Smith & Hipp, 1994; Kamm, 1995) inter alia, the design of the dialogue manager in real-time, implemented systems is still more of an art than a science (Sparck-Jones & Galliers, 1996). This chapter discusses a method, and experiments that validate the method, by which a spoken dialogue system can learn from its experience with human users to optimize its choice of dialogue strategy.

The dialogue manager of a spoken dialogue system accepts the user’s utterance which is represented as a frame of Spoken Language Understanding modules results and then chooses in real time what information to communicate to the human user at a conceptual level and how to communicate it. The choice it makes is called its ‘strategy’ The system responses have to reflect the discourse context by maintaining the discourse history.

The dialogue manager can be formulated as a state machine, where the state of the dialogue is defined by a set of state variables representing observations of the user’s conversational behaviour, the results of accessing various information databases, and aspects of the dialogue history. Transitions between states are driven by the system’s dialogue strategy. However, There are a many possible choices for policies at each state of a dialogue. Decision theoretic planning can be applied to the problem of choosing among dialogue strategies, by associating a utility U with each strategy (action) choice and by positing that spoken dialogue systems should adhere to the ‘Maximum Expected Utility Principle’ which states that an optimal action is one that max-imizerizes the expected utility of outcome states (Keeney & Raiffa, 1976; Russell & Norvig, 1995), Thus, a SDS can act optimally by choosing a strategy (a) in state
3. DIALOGUE, REPRESENTATION, INFERENCE AND LEARNING STRATEGIES

$S_i$ that maximizes $U(S_i)$. Several reinforcement learning algorithms based on dynamic programming specify a way to calculate $(S_i)$ in terms of the utility of a successor state $S_j$ (Bellman, 1957; Watkins, 1989; Sutton, 1991; Barto, Bradtke, & Singh, 1995), so if the utility for the final state of the dialogue were known, it would be possible to calculate the utilities for all the earlier states, and thus determine a policy which selects only optimal dialogue strategies. Previous work suggested that it should be possible to treat dialogue strategy selection as a stochastic optimization problem in this way (Walker, 1993; Biermann & Long, 1996; Levin, Pieraccini, & Eckert, 1997; Mellish, Knott, Oberlander, & O’Donnell, 1998). There are three main possibilities for a simple reward function: user satisfaction, task completion, or some measure of user effort such as elapsed time for the dialogue or the number of user turns. But it appeared that any of these simple reward functions on their own fail to capture essential aspects of the system’s performance. For example, the level of user effort to complete a dialogue task is system, domain and task dependent. Moreover, high levels of effort, e.g., the requirement that users confirm the system’s understanding of each utterance, do not necessarily lead to concomitant increases in task completion, but do lead to significant decreases in user satisfaction (Shriberg, Wade, & Price, 1992; Danieli & Gerbino, 1995; Kamm, 1995; Baggia et al., 1998). Furthermore, user satisfaction alone fails to reflect the fact that the system will not be successful unless it helps the user complete a task. A method for deriving an appropriate performance function was a necessary precursor to applying stochastic optimization algorithms to spoken dialogue systems in the paradise method for learning a performance function. In this chapter, we apply the paradise model (Walker, Litman, Kamm, & Abella, 1997a) to learn a performance function from a corpus of human-computer dialogues, which we then use for calculating the utility of the final state of a dialogue in experiments applying reinforcement learning to selection of dialogue strategies.

3.1 Dialogue State Representation

In a spoken dialogue system, the dialogue manager form the heart of the system. After the utterance is converted into a natural language form and the meaning is interpreted by natural language module, the dialogue manager has to decide how and what to say to the user to fulfil the system’s turn in the dialogue. In order to obtain
computationally efficient algorithms, the structure of the domain under consideration must be exploited. The Dialogue Manager in the HBIS has to maintain the state which is defined by a set of state variables that represent the information that has happened during the dialogue process and aids the dialogue manager in deciding what the system should do in opposition to the user utterances. The state variables encode various observations of the user conversational behaviour, such as results of processing speech with the natural language understanding module and results from accessing information databases relevant to the application as well as certain aspects of the current context. The better way to represent the state is using a probabilistic approach as a belief distribution over environment states and updated using Bayes theorem.

### 3.1.1 Probabilistic Graphical Models

Probabilistic Graphical Models are graphs in which nodes represent the random variables and the arcs represent the conditional independence assumptions and thus provide a representation for the joint probability distributions. A graphical model needs fewer parameters based on the conditional assumptions and thus fit for efficient inferencing and learning when compared to other representations.

There are three types of graphical models:

1. **Directed Graphical Models** also known as Bayesian Networks, Belief Networks, Generative models, Casual models etc are graphs in which the arc are directed and are mostly used in machine learning applications.

2. **Un-Directed Graphical Models** also known as Markov networks, Markov Random Fields are graphs in which the arc are undirected.

3. **Chain Graphs** are models in which the arc are both directed and undirected.

### 3.1.2 Bayesian Networks

It’s a directed acyclic graphical model which give an intuitive representation for the various assumptions and belief states in a system and also facilitate the use of computationally effective algorithms for updating the beliefs in an environment state whenever an observation is made. A Bayesian Network is a representation for statistical models where each node represents a random variable and the edges represent the probabilistic
constraints between edges. If an arc between two nodes X and Y is interpreted as "X causes Y". The joint distribution of all variables in the graph factorises as the product of the conditional probability of each variable given its parents in the graph. In a POMDP based framework the assumptions are represented by the network as shown in the figure 3.1

![Figure 3.1: A portion of Bayesian network representing the POMDP Model](image)

Networks as shown in the figure which repeat the structure (time-slices) at each interval in time are refereed as Dynamic Bayesian Networks. Actions of the system ($a_t$) are shown in the rectangles and Shading of Observation nodes ($O_t$) represent that they are observed. The Bayesian networks for dialogue allow further factorization of the dialogue system environment state ($S_t$) which aids the updation of factor beliefs using various efficient algorithms. [Williams and Young, 2005] factorized the environment state into three components $s_t = (g_t, u_t, h_t)$ where $g_t$ represents the long term goal of the user, $u_t$ represents the true user act and $h_t$ represents the dialogue history. Further structuring can be done by representing the state into slots $c \in C$ where slot implies a concept for which the user must specify a value e.g. In a TIS the concept may be destination or type of accommodation or food. The state is thus factorised into sub-goals $g_t^{(c)}$ or sub-histories $h_t^{(c)}$. The sub-histories $h_t^{(c)}$ depend on the user act $U_t$ and the previous sub-histories $h_{t-1}^{(c)}$. The user act depends on the set of sub-goals $g_t^{(c)}$. 

39
3.1.2.1 Dependencies

The exists a strong dependency between the concepts of the real world dialogue as the user intention clarifies. Given the user action, the sub-goal nodes cannot be assumed to be independent as such there will be a dependency which is to be limited to enable tractability whereas the sub-history nodes can be independent. A method to limit the dependencies is to add the validity node \([v^c_t]\) for each concept which indicates whether the associated sub-goal is relevant to the overall user goal or not. The validity node takes the value either ‘Applicable’ or ‘Not Applicable’. If the validity node is ‘Applicable’ then the sub-goal also become applicable and the user sub-goal depends on the previous value with some probability of change.

Figure 3.2: Factorisation of Bayesian network representing part of a health based information system - \(v_{\text{disease}}\) is the validity node for disease, \(g_{\text{type}}\) represents the type of department being sought.

Figure 3.2 shows the network for a health based information system representing two types of concept viz. the type of department the patient wants to visit and the disease the patient is suffering from. The user act and user goal are assumed to be independent from the previous history. When a patient asks for the type of department, the disease concept may not be applicable. But once the patients talks about the
disease it becomes relevant and hence applicable. Thus it clearly indicate the intention of the user that he wants to visit a particular department for a disease he mentions and hence the validity of the disease node increases.

### 3.1.2.2 History Nodes

History nodes store the information about the acts that have happened in a dialogue and is used for framing a dialogue strategy. The sub-history is separated into the what the user wants/desires to know $d_t^{(c)}$ and the grounding information for each concept $i_t^{(c)}$. The desire variable $d_t^{(c)}$ may requires only few values such as NOTHING SAID, REQUESTED, INFORMED. This allows the system to record when the user requests for the value of a concept. The grounding information nodes $i_t^{(c)}$ stores the last grounding state for the concept value.

### 3.1.2.3 User Acts and Observation Nodes

In a real world spoken dialogue system there are large number of state variables and updating the belief states of these variables in a Bayesian network will be computationally expensive. In such situations, the user acts are split for each concept represented as $u_t^{(c)}$ and depends on the user goal for that concept. Similarly the observation nodes are split into sub-observation $o_t^{(c)}$ which store how a observation is related to a given concept. Actions that do not apply to any concept appear in all concept-level observations.

<table>
<thead>
<tr>
<th></th>
<th>Confidence Score</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overall Observation</strong></td>
<td>inform(type=orthopaedics, disease=fracture)</td>
</tr>
<tr>
<td>$o_t$</td>
<td>inform(type=orthopaedics, disease=pain)</td>
</tr>
<tr>
<td><strong>Type Observation</strong> $o_{type}$</td>
<td>inform(type=orthopaedics)</td>
</tr>
<tr>
<td><strong>Disease Observation</strong> $o_{disease}$</td>
<td>disease=fracture</td>
</tr>
<tr>
<td></td>
<td>disease=pain</td>
</tr>
</tbody>
</table>

**Table 3.1: Split of overall observation into concept level observations**

Since the observations are completely factorised, the approach may not give better performance. However this approach can be used for modelling highly complex user action models with the help of independent concept level user acts for which different
user action probabilities are used for each concept which are then joined to determine the probability of the overall user action.

![Figure 3.3: Bayesian network with splitted user-acts.](image)

### 3.2 Factor Graphs

Factor Graphs (fgraphs) is a representation which unifies directed and undirected graphs [Kschischang et al., 2001] proposed a graphical framework known as *Factor graphs* which provides a suitable formalism for analysing the independence of variables in a spoken dialogue system. Also there exists many efficient algorithms for updating the beliefs which can be used by the dialogue manager. Factor graphs are undirected bipartite graphs comprising of two types of nodes which represent random variables (Circle nodes) and factors (Square node). Factor Graphs are bipartite because each
variable node $X_i$ is connected to all the factor nodes $F_i$ which contain $X_i$ in their domains.

In general, factor graphs specify how a function of many variables can be decomposed into a set of local functions. Factors are not probabilities themselves, they are functions that determine all probabilities. The joint distribution over all random variables can be written as a product of factor functions, one for each factor node. These factors are a function of only the random variables connected to the factor node in the graph. There is a direct mapping from Bayesian networks to factor graphs. Figure 3.4 is a factor graph representation of the POMDP assumptions, previously depicted as a Bayesian network in Figure 3.1.

![Figure 3.4: Factor graph representing POMDP model -](image)

In this factor graph, the environment state transition function is represented as $f^{(\text{trans})}_t$ and thus $f^{(\text{trans})}_t(s_t, s_{t+1}, a_t) = P(s_{t+1} | s_t, a_t)$. Also $f^{(\text{obs})}_t$ represents the observation function $f^{(\text{obs})}_t(s_t, o_t, a_{t+1}) = P(o_t | s_t, a_{t-1})$. The variables in the factor graph are denoted by $X_i$ and the variable values by $x_i$, the factors by $f_\beta$ and vector $x = (x_1, x_2, ... x_{N_i})$ represent the variable values simultaneously. Each factor will depend on a subset of random variables. As such they can be defined as functions over the whole set denoted by $f_\beta(x)$ which is achieved by defining factor values as factor values of the original subset. Thus the joint distribution in the factor graph factorises as

$$p(x) \propto \prod_\beta f_\beta(x)$$

(3.1)
3. DIALOGUE, REPRESENTATION, INFERENCE AND LEARNING STRATEGIES

3.3 Loopy Belief Propagation Algorithm

3.3.1 Belief Propagation

Belief propagation is a way of computing exact marginal posterior probabilities in graphs with no undirected cycles (loops). The system takes an action based on the current set of beliefs in the environment state which must be updated whenever an observation is made. The decision can be based on the marginal distribution of the beliefs over a single variable and is computed by integrating or summing out all other random variables from the joint distribution. The marginal distribution provides the information that can aid the dialogue management decision and thus save the computational time. The \( \tilde{x}_1 \) notation indicates that the variable \( X_1 \) is fixed while the other variables are either integrated or summed thus the marginal \( p(\tilde{x}_1) \) would be computed as

\[
p(\tilde{x}_1) = \int p(\tilde{x}_1, x_2,\ldots, x_N) \, dx_2,\ldots, dx_N.
\]

\[
= \sum_{x_2,\ldots,x_N} p(\tilde{x}_1, x_2,\ldots, x_N)
\]

\[
= \sum_{x: x_1 = \tilde{x}_1} p(x)
\]

Belief Propagation provides exact inferencing when there are no loops in graph (e.g. chain, tree.) It is equivalent to dynamic programming/Viterbi in these cases. The marginal distribution is generally not tractable as the belief propagation becomes exponential in the size of the nodes. So an approximate algorithm like the Loopy Belief Propagation (LBP) or sum-product algorithm [Kschischang et al., 2001] is to be used to enable tractability.

When loops are present in the network the network is no longer single connected. The local propagation schemes may not work and will run into trouble. If the loops are ignored, and permit the nodes to communicate with the factors, the message will circulate around the loops and the process may not converge to a stable equilibrium. Loopy propagation algorithm maintains a set of messages for each arc in the model. For each arc between a node representing a random variable \( X_i \) and a factor \( f_a \) there are two defined messages. \( \mu_{X_i \rightarrow f_a}(x_i) \) is a message from the variable to the factor and \( \mu_{f_a \rightarrow X_i}(x_i) \) is the message from the factor to the random variable. Both of these are the functions of the possible values of \( X_i \). Once the messages are computed the marginal
probability of a random variable $X_i$ is calculated from the message to that variable from the neighbouring factors, $a \in ne(X_i)$ If $k$ is the normalizing constant then

$$p(x_i) = k \prod_{a \in ne(X_i)} \mu_{f_a \rightarrow X_i}(x_i)$$  \hspace{1cm} (3.2)

**Algorithm 3.1** Loopy Belief Propagation Algorithm

**Initialize**: Set all messages equal to one.

Let $Y = \{x = (x_1, x_2, ..., x_N)^T | x_i$ is the possible value of $X_i\}$

repeat

Choose a factor $f_a$ to update. Suppose this is connected to variables $X_1, X_2, ..., X_N$.

First update the approximation as follows:

for each variable $X_i$ connected to the factor do

Update the message out of the factor

$$\mu_{f_a \rightarrow X_i}(x'_i) = \sum_{x \in Y, x_i = x'_i} \prod_{j \neq i} \mu_{X_j \rightarrow f_a}(x_j).$$

Update the cavity distributions

Update the message into nearby factors

for each factor $b \neq a$ connected to variable $X_i$ do

$$\mu_{X_i \rightarrow f_b}(x'_i) = \prod_{a \neq b} \mu_{f_a \rightarrow X_i}(x'_i).$$

end for

end for

until convergence

The iterative process of belief updates using factor graphs continues until the approximate distribution no longer changes significantly, at which stage the algorithm is said to be converged and the resulting set of messages will constitute a fixed point of the algorithm. If the factor graph has a tree structure, the algorithm will converge after a breadth first traversal and followed by a reverse sequence of updates.

### 3.4 Limitations

In a dynamic bayesian network like POMDP the number of nodes grows with time. In order to update its beliefs the system will have to maintain all the information for all the nodes in the network i.e. for the most recent time-slice, the approximations for
3. DIALOGUE, REPRESENTATION, INFERENCE AND LEARNING STRATEGIES

all the previous time-slices are needed. This issue contradicts the MArkov property of POMDP i.e. the beliefs of the current time slice depends on the beliefs at the previous time-slice. The Loopy Belief Propagation algorithm computes approximate marginal distributions and if the network is highly connected, the observations from future time-slices may effect the computation of marginals of the previous time-slices which then affects the computation of marginals at the current time-slice. Thus it will be a added responsibility for the system to store the information related to the cavity and factor approximation for all nodes for the entire duration of the dialogue. To overcome the difficulty of recomputing the approximation for all previous time-slices for the lengthy dialogue, it is preferable to limit the number of time-slices that are maintained [Murphy, 2002] Given a number $n$, the factor updates are limited to $n$ time-slices, the marginals for variables connected to factor nodes are computed at time $t$ by approximating the joint distribution at time $t-1$ for the most recent $n$ time-slices and are maintained for future updates. [Boyen and Koller, 1998] suggested an approach wherein only the marginal approximation at time $t-1$ is used to compute the exact marginal distributions of the current time slice which are then stored and used for the next iteration.

3.5 Expectation propagation

Complex Spoken Dialogue System have to deal with a large state spaces and when the nodes have a large number of values it becomes difficult to update the beliefs using Loopy Belief Propagation Algorithm. In case of arbitrary approximation, the marginal matching can be replaced by minimizing a distance function between two probability distribution known as divergence measure. Expectation propagation [Minka, 2001] is like belief propagation except it requires that the posteriors (beliefs) on each variable have a restricted form. Specifically, the posterior must be in the exponential family of the form $q(x) \propto \exp(\gamma f(x))$ where $x$ is a variable. This ensures that beliefs can be represented using a fixed number of sufficient statistics. We choose the parameters of the beliefs such that

$$\gamma^* = \arg \min \, D(p(x) \parallel q_\gamma(x))$$

where $p(x)$ and $q(x)$ are two functions for which the distance measure is defined.

$$p(x) = \frac{q^{\text{prior}}(x) \times t(x)}{Z}$$
3. DIALOGUE, REPRESENTATION, INFERENCE AND LEARNING STRATEGIES

is the exact posterior and \( Z = \int x q_{\text{prior}}(x) \times t(x) \) is the exact normalizing constant, \( t(x) \) is the likelihood term (message come in from a factor and \( q_x(x) \) is the approximate posterior. If the Sequential Bayesian Updation is combined with this approximation after every update step, it is known as Assumed Density Filtering (ADF), or the probabilistic editor which depends on the order in which the update are made on the beliefs. The Expectation Propagation Algorithm being a batch algorithm reduces the sensitivity to ordering by iterations wherein it goes back and re-optimizes each belief in the revised context of the updated beliefs. To achieve this all the messages are to be stored for undoing any of their effect. Thus instead of approximate matching of messages, the posterior are matched using moment matching. Consider the factor graph given in fig 3.6 to understand the difference between the Belief Propagation and Expectation Propagation. A message is sent from \( f \) to \( x \) and then \( x \) belief’s are updated as

\[
\phi^\text{prior}_x = \frac{\phi_x}{\mu^\text{old}_{f \rightarrow x}} = \frac{\mu^\text{old}_{g \rightarrow x}}{
\phi^\text{prior}_f = \frac{\phi_f}{\mu^\text{old}_{x \rightarrow f}} = f(x,y)\mu^\text{old}_{y \rightarrow f}(y)
\mu_f \rightarrow x = \phi^\text{prior}_f \downarrow x = \int_y f(x,y)\mu^\text{old}_{y \rightarrow f}(y)
\phi_x = \phi^\text{prior}_x \times \mu_f \rightarrow x = \mu^\text{old}_{g \rightarrow x} \times \mu_f \rightarrow x
\]

In Expectation Propagation, the approximate posterior \( \phi_x \) is computed first and then the message \( \mu_f \rightarrow x \) is derived which if combined with the prior \( \phi^\text{prior}_x \) would result in the same approximate posterior.

\[
\phi^\text{prior}_x = \frac{\phi^\text{old}_x}{\mu^\text{old}_{f \rightarrow x}}
\]
3. DIALOGUE, REPRESENTATION, INFERENCE AND LEARNING STRATEGIES

\[ \phi_f^{\text{prior}} = \phi_f / \mu_{x \rightarrow f} \]

\[ (\phi_x, Z) = ADF(\phi_x^{\text{prior}} \times \phi_f^{\text{prior}} \downarrow x) \]

where \((q, Z) = ADF(p)\) produces the best approximation from \(q\) to \(p\) within a specified family of distributions.

**Algorithm 3.2 Expectation Propagation Algorithm**

For each factor \(f\) in order.

\( \phi_f = \phi_f^{\text{old}} \)

for each variable \(x\) connected to the factor \(f\) in predecessor order do

Update the message out of the factor

\[ \mu_{x \rightarrow f} = (\phi_x^{\text{old}})(\mu_{f \rightarrow x}^{\text{old}}) \]

\[ \phi_f = \phi_f \times (\mu_{x \rightarrow f}^{\text{old}})/(\mu_{x \rightarrow f}^{\text{old}}) \]

end for

for each variable \(x\) connected to the factor \(f\) in successor order do

Update the message in to the factor

\[ \phi_x^{\text{prior}} = \phi_x^{\text{old}} / \mu_{x \rightarrow f}^{\text{old}} \]

\[ \phi_f^{\text{prior}} = \phi_f / \mu_{x \rightarrow f}^{\text{old}} \]

\[ (\phi_x, Z) = ADF(\phi_x^{\text{prior}} \times \phi_f^{\text{prior}} \downarrow x) \]

\[ \mu_{f \rightarrow x} = (Z\phi_x)(\phi_x^{\text{prior}}) \]

end for

3.6 Comparison to previous work

The key feature of the Loopy Belief Propagation is that dialogue manager can deal with complex dependencies between variables by using an approximate updates instead of exact updates. Past research work of [Bui et al., 2009] which is close to LBP assumes a completely independent factorization of goals and can therefore use the standard Loopy belief Propagation to obtain an exact update. [Young et al., 2010] suggested the Hidden Information State approach which updates probabilities by partitioning the unfactorized state space into group of states which are indistinguishable given the observations. It can be shown that the probabilities for all states in a given partition may be updated simultaneously if the user goal never changes. The use of Loopy Belief
Propagation allows to reduce the computation times by exploiting conditional independence and allow probabilities that the user goal may change. [Henderson and Lemon, 2008] provided an alternative mechanism for state update similar to HIS update based on the Markov Decision Process state mixture. Another approach for belief updates is suggested by [Williams, 2007] and based on particle filters which uses sampling. The approach has shown considerable benefits in terms of computation time of updates when compared to exact updates.

3.7 Grouped Loopy Belief Propagation

Belief updates when the concept takes multiple values improves the computational complexity of the dialogue system. Though Loopy Belief Propagation algorithm exploits the dependencies between the concepts but updating beliefs becomes complex when the concept takes a large number of values e.g the disease concept can take multiple values in the environment state. The solution to this problem was proposed by [Young et al., 2010] and it suggested to join the indistinguishable environment states into groups.

3.8 Policy Design and Learning

After the system’s belief state is defined, the dialogue policy or strategy $\pi$ which means how the actions are taken is to be defined. In case of Spoken Dialogue system we propose to use reinforcement learning to optimize the policy in which the reward function is defined by the function $r(b, a)$ which indicates the reward obtained by taking the action $a$ when the system is in belief state $b$. But the system should take action which will maximize the total expected reward in a dialogue which is based on the assumption that the belief state transitions are directed and depend on the previous value of $b$. The total expected reward is computed as

$$E(R) = E(\sum_{t=1}^{T} r(b_t, a_t)) \tag{3.3}$$

In POMDP, the belief state are probability distribution and hence continuous. Thus $p(b' | b, a)$ denotes the probability density function. The expected future reward
when starting in belief state \( b \) and following the dialogue policy \( \pi \) is recursively given by

\[
V^\pi(b) = \sum_a \pi(b, a)r(b, a) + \sum_a \int_{b'} \pi(b, a)p(b'|b, a)V^\pi(b')
\]

When Working with Markov Decision Processes various other functions help in the task of policy optimization by choosing one of the policy(\( \pi \)) which maximizes \( V^\pi(b_0) \) where \( b_0 \) is the start belief start e.g.

The \textit{Q-function} \( Q^*(b, a) \) is the expected future reward obtained by starting with a particular action and then following the policy and is given by

\[
Q^*(b, a) = r(b, a) + \int_{b'} p(b'|b, a)V^\pi(b')
\]

The \textit{advantage function} \( A^*(b, a) \) is the difference between the Q-function and the total expected reward or value function \( V^\pi \) and is given by

\[
A^*(b, a) = Q^*(b, a) - V^\pi(b)
\]

The \textit{occupancy frequency} \( d^*(b) \) gives the expected number of times each state is visited.
In the given equation \( p(b_t = b) \) denotes a probability density and \( d^*(b) \) is a form of density function.

\[
d^*(b) = \sum_{t=0}^{\infty} p(b_t = b)
\]

In a dialogue system, there are situations wherein it is clear to the system designer that taking a particular action is better than the others. \textit{Summary actions} thus reduce the size of the action set during the policy learning and thus enable the system designer to embed the expert knowledge to allow learning to be quick and efficient. So along with machine actions \( \overline{A} \) the summary actions \( A \) which are a subset of the machine actions are defined which are used for learning. Given a summary action \( a \) and a belief state \( b \) a mapping back to the original action set \( F(a, b) \) is also defined. The use of summary actions is based on the summary POMDP idea proposed by [Williams and Young, 2005] which factors the state and actions according to a set of concepts.

3.8.1 Function approximation

The dialogue system can always reach a belief state which has never been observed earlier. So function approximation are to be used to generalize the past experiences to
new belief states and depend on the actions as well as beliefs. The standard approach used is linear function approximation for either the value function $V$, the Q-function or the policy $\pi$ in which the approximation is parameterised by a vector, $\theta$, where the entries of $\theta$ are called policy parameters. The summary features which are computed from the belief states as a summary of the important characteristics are also used for function approximation and compiled into a vector $\phi(b)$. Features can include the entropy of the concept or the most likely probability for a concept. Given a set of features, different parameters will be used in the approximation, depending on which the action is taken. If the policy parameters for action $a$ are denoted by $\theta_a$ then the approximation for the Q-function would be

$$Q(a, b, \theta) \approx \theta_a \cdot \phi(b) \quad (3.8)$$

Some of the parameters from different actions are tied by the use of basis function $\phi_a(b)$ for optimization. Thus the approximation for the Q-function is given by

$$Q(b, a, \theta) \approx \theta \cdot \phi_a(b) \quad (3.9)$$

### 3.9 Natural Actor Critic Algorithm

The algorithm Natural Actor Critic Algorithm [Peters and Schaal, 2008] is a modified form of gradient descent machine learning algorithm which we used in the framework and aided the spoken dialogue system to learn the parameters that optimize the expected future reward after the policy was parameterised using a suitable structure. Various other alternative algorithms e.g Temporal Difference Learning [Sutton and Barto, 1998] and Least Squares Temporal Difference Learning [Bradtke and Barto, 1996] have shown large fluctuations in policy performance during learning. The gradient descent algorithm uses the Euclidean matric as a measure of distance and iteratively subtracts a multiple of the gradient from the parameters being estimated. In general, the parameter space is known as Riemann space and for small changes in the parameters $\theta$, a metric tensor, $G_{\theta}$ is defined such that the distance is $|d(\theta)|^2 = d\theta^T G_{\theta} d\theta$. [Amari, 1998] showed that for optimizing an arbitrary loss function in a general Riemann space the direction of the steepest descent also known as natural gradient as compared the traditional vanilla descent is given by $G^{-1}_{\theta} \cdot \nabla_{\theta} L(\theta)$. [Amari, 1998] also showed the optimal
metric tensor which gives distances that are invariable to scale with the parameters is the Fisher Information Matrix $G_\theta$ which is given by

$$ (G_\theta)_{ij} = E\left( \frac{\partial \log p(x|\theta)}{\partial \theta_i} \frac{\partial \log p(x|\theta)}{\partial \theta_j} \right) $$

where $p(x|\theta)$ is a given probability distribution. [Peters and Schaal, 2008] showed that the Fisher Information Matrix for a Markov Decision Process is given by

$$ G_\theta = \int_B d^\pi(b) \int_A \pi(a|b,\theta) \nabla_\theta \log \pi(a|b,\theta) \nabla_\theta \log \pi(a|b,\theta)^T da db. $$

The direction of the steepest descent is the inverse of this matrix multiplied by the vanilla gradient which is given by Policy Gradient Theorem as

$$ \nabla_\theta V(b_0, \theta) = \int_B d^\pi(b) \int_A A^\pi(b,a) \pi(a|b,\theta) \nabla_\theta \log \pi(a|b,\theta) da db $$

The equation depends on the advantage function and the occupancy frequency where the advantage function is approximated and the integral over the occupancy frequency is approximating using sampling methods where the rewards in the dialogue are grouped together to obtain suitable estimates. The sum of rewards gives an unbiased estimate of the sum of advantages and initial value function. If the approximate advantage function $\hat{A}_w(b, a)$ is chosen such that

$$ \hat{A}_w(b, a) = \nabla_\theta \log \pi(a|b,\theta) . w $$

where $w$ minimises the average squared approximation error i.e.

$$ \frac{\partial}{\partial w} \int_B d^\pi(b) \int_A \pi(a|b,\theta) (A^\theta(b,m) - \hat{A}_w(b,a))^2 da db = 0 $$

then the required natural gradient is given by

$$ G_\theta^{-1} \nabla_\theta V(b_0, \theta) = w. $$

The gradient has been used in the algorithm which is known as Natural Critic Algorithm that iterates between the evaluation step also known as critic step wherein the approximate advantage function is estimated and improving step wherein the actor improvement is done by changing the parameters by a multiple of natural gradient. The algorithm is sure to converge to a local maximum of the value function if the requirements are satisfied.
Algorithm 3.3 Natural Actor Critic Algorithm

for each dialogue, $n$ do

   Execute the dialogue according to the current policy $\pi$
   Obtain the sequence of states $b_{n,t}$ and machine actions $a_{n,t}$
   Compute the statistics for the dialogue

   $\psi_n = \left[ \sum_{t=0}^{T_n} \nabla \log \pi(a_{n,t}|b_{n,t}, \theta)^T, 1 \right]^T$

   $R_n = \sum_{t=0}^{T_n} r(b_{n,t}|a_{n,t})$

Critic Evaluation
Choose $w$ to minimize $\sum_n (\psi_n^T w - R_n)^2$

Actor Improvement
Update the policy parameters
$\theta_{n+1} = \theta_n + w_0$ where $w^T = [w_0^T, J]$.
Propagate the impact and deweight the previous dialogue’s
$R_i \leftarrow \gamma R_i$, $\psi_i \leftarrow \gamma \psi_i$ for all $i \leq n$
end for
3. DIALOGUE, REPRESENTATION, INFERENCE AND LEARNING STRATEGIES

3.10 Evaluation

To evaluate the algorithm and the optimization of dialogues, a large number of human users dialogues are required which is prohibitive. Instead a simulator of the environment of the dialogue system is built and the system’s policy can be optimized by interacting with the simulator instead of the human users. Optimum policies which are trained on the simulator are then used to bootstrap a policy which are further trained by interacting with the human user in the real world. The simulator includes both the user simulator, i.e. how the user behaves and responds when using the system and the error simulator i.e. how confusions are generated.

![Figure 3.6: Plot showing the trend in the reward during policy training with a sample 100 dialogues -](image)

3.11 Conclusion

The chapter has discussed the various methods to model and update the various states in a dialogue process. Bayesian Networks is an efficient and effective structure for
modelling the various states in a spoken dialogue system which can be used by the
dialogue manager for effective responding during the system’s turn as its beliefs are
updated with an approximated posteriori marginal distribution by the Loopy Belief
Propagation algorithm which makes the system tractable by exploiting the conditional
independence assumptions and limiting the time-slices for which the approximations
are made in the factor graph. The chapter also shows how Natural Critic Algorithm
can be used to learn the policy which tends to converge optimally. It has also been
checked that this framework work well with human users as well as simulators.