Chapter 3

Probabilistic models of dialogue

The previous chapter provided an overview of some of the most influential approaches to dialogue management, and outlined in particular the benefits of statistical techniques to account for the uncertainty and unpredictability inherent to spoken dialogue. The present chapter provides further background on the theoretical and methodological foundations of these statistical approaches as well as their application to the dialogue management task.

The first section of this chapter concentrates on the use of graphical models to design structured representations of probability and utility distributions. Graphical models provide mathematically principled methods for representing, estimating and reasoning over complex probabilistic domains. The section starts with a well-known type of directed graphical model, namely Bayesian networks. We then show how to extend Bayesian networks to capture temporal sequences and express decision-theoretic problems through actions and utilities. We also review the most important inference and learning algorithms developed for these graphical models.

The second section presents the fundamental principles of reinforcement learning, which is the learning framework employed by most recent statistical approaches to dialogue management. The section starts with a formal definition of a Markov Decision Process and explains how policies can be automatically optimised for such a process. The discussion is finally extended to partially observable environments in which the current state is hidden and must be indirectly inferred from observations.

Once the mathematical foundations of graphical models and reinforcement learning are in place, the third and last section of this chapter describes how these concepts and techniques can be practically applied to model dialogue management tasks. The section provides a survey of the multiple approaches that have been developed in the last two decades to automatically optimise dialogue policies based on various flavours of supervised and reinforcement learning.

3.1 Graphical models

We describe in this section the core properties of (directed) graphical models,[1] their formal representation, and their use in learning and inference tasks.

[1] A variety of undirected or partially directed graphical models also exist, amongst which we find Markov networks and Conditional Random Fields, but they will not be discussed nor employed in this thesis.

3.1.1 Representations

Bayesian networks

Let $\mathbf{X} = X_1 \ldots, X_n$ denote a collection of random variables, where each variable X_i is associated with a range of possible values. This range can be either discrete or continuous. For dialogue models, the range of a variable X_i is typically discrete and can be explicitly enumerated. The enumeration of values for the variable X_i can be written $Val(X_i) = \{x_i^1, \ldots, x_i^m\}$.

In the general case, the variables $\mathbf{X}$ can be interrelated by complex probabilistic dependencies. These dependencies can be expressed through the joint probability distribution $P(X_1, \ldots, X_n)$. The size of this joint distribution is, however, exponential in the number n of variables, and is therefore difficult to manipulate (let alone estimate and reason over) directly.

We can fortunately exploit conditional independence properties to reduce the complexity of the joint probability distribution. For three disjoint sets of random variables $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$, we say that $\mathbf{X}$ and $\mathbf{Y}$ are conditionally independent given $\mathbf{Z}$ iff $P(\mathbf{X}, \mathbf{Y} \mid \mathbf{Z}) = P(\mathbf{X} \mid \mathbf{Z})P(\mathbf{Y} \mid \mathbf{Z})$ for all combinations of values for $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$. This conditional independence is denoted $(\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z})$.

Conditional independence allows a joint probability distribution to be decomposed into smaller distributions that are much easier to work with. For a variable X_i in $X_1, \ldots, X_n$, we can define the set $parents(X_i)$ as the minimal set of predecessors[2] of X_i such that the other predecessors of X_i are conditionally independent of X_i given $parents(X_i)$. The set of parents can be empty if the variable X_i is independent of all other variables. This definition enables us to decompose the joint distribution based on the chain rule:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid X_1, \ldots, X_{i-1}) \tag{3.1}$$

$$= \prod_{i=1}^{n} P(X_i \mid parents(X_i)) \tag{3.2}$$

This decomposition can be graphically represented in a *Bayesian network*. A Bayesian network is a directed acyclic graph (DAG) where each random variable is represented by a distinct node. These nodes are connected via directed edges that reflect conditional dependencies. In other words, an edge $X_m \rightarrow X_n$ indicates that $X_m \in parents(X_n)$. Each variable X_i in the Bayesian network must be associated with a specific conditional probability distribution $P(X_i \mid parents(X_i))$. Together with the directed graph, the conditional probability distributions (abbreviated as CPDs) fully determine the joint probability distribution of the Bayesian network.

Given such definitions, the Bayesian network can be directly used for inference by querying the distribution of a subset of variables, often given some additional evidence. Two operations are especially useful when manipulating probability distributions:

- Marginalisation (also called "summing out"), which derives the probability of the variable X given its conditional distribution $P(X \mid Y)$ and the distribution $P(Y)$:

$$P(X) = \sum_{y \in Val(Y)} P(X, Y = y) = \sum_{y \in Val(Y)} P(X \mid Y = y)P(Y = y) \tag{3.3}$$

[2]The predecessors of X_i are simply the variables $X_1, \ldots, X_{i-1}$ (under a particular ordering for the variables).

- Bayes' rule, which reverses the order of a conditional distribution between two variables X and Y (possibly with some background evidence $\mathbf{e}$):

$$P(X \mid Y, \mathbf{e}) = \frac{P(Y \mid X, \mathbf{e}) P(X \mid \mathbf{e})}{P(Y \mid \mathbf{e})} \tag{3.4}$$

As an illustration, Figure 3.1 provides an example of a Bayesian network that models the probability of occurrence of a fire at a given time. The probability of this event is dependent on the current weather. In addition, two (imperfect) monitoring systems are used to detect possible fires; one on the ground, and one via satellite.

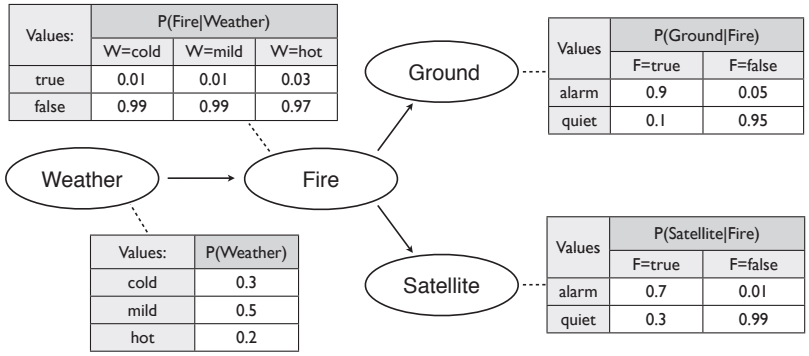


Figure 3.1: Example of a Bayesian network with four random variables.

For each random variable X_i, there is one distinct distribution for every combination of values in $parents(X_i)$. The probabilistic model defined in the figure includes therefore a total of eight distributions (corresponding to the eight columns). The distributions in Figure 3.1 are called *categorical* distributions.[3] Categorical distributions can be implemented with look-up tables that map every possible value in $Val(X_i)$ to a particular probability. Many of the probability distributions used throughout this thesis will take the form of categorical distributions. Other representations for discrete CPDs are conceivable, as will be shown later in this thesis.

A Bayesian network can also contain continuous probability distributions. These distributions are usually encoded with *density functions* represented in a parametric form. A well-known parametric distribution is the normal distribution $\mathcal{N}(\mu, \sigma^2)$, which is defined by its two parameters μ (the mean) and σ^2 (the variance). Continuous distributions can alternatively be expressed with non-parametric methods based on e.g. kernel density estimators (see e.g. Bishop, 2006; Koller and Friedman, 2009, for more details on these continuous distributions).

Appendix A enumerates the most important discrete and continuous probability distributions employed in this thesis.

[3]Categorical distributions are often conflated with *multinomial* distributions, which specify the number of times an event will occur in a repeated independent trial with k exclusive categories, with each category having a fixed probability. A categorical distribution is equivalent to a multinomial distribution for a single observation.

Reasoning over time

In order to apply Bayesian networks to dialogue management, two additional elements are necessary. The first extension is to allow variables to evolve as a function of time. Such temporal dependencies are indeed necessary to account for the dynamic nature of dialogue (the dialogue state is not a static entity). Two assumptions are usually made to capture temporal sequences:[4]

1. The first assumption, called the Markov assumption, stipulates that the variable values at time t only depend on the previous time slice $t-1$. Formally, let $\mathbf{X}$ be an arbitrary collection of variables. We denote by $\mathbf{X}_t$ the random variables that express their values at time t. The Markov assumption states that $(\mathbf{X}_{t+1} \perp \mathbf{X}_{0:(t-1)} \mid \mathbf{X}_t)$.

2. The second assumption is that the process is stationary, which means that the probability $P(\mathbf{X}_t \mid \mathbf{X}_{t-1})$ is identical for all values of t. A *stationary* process must be distinguished from a *static* process: A static process is a stochastic process that remains constant for all time steps. In contrast, a stationary process can change over time, but the transition model that describes the dynamics of this process remains constant.

Given these two assumptions, we can define a stochastic process with a probability distribution $P(\mathbf{X}_t \mid \mathbf{X}_{t-1})$ that specifies the distribution of the variables $\mathbf{X}$ at time t given their values at time $t-1$. Such a model is called a *dynamic Bayesian network* (DBN). The distribution $P(\mathbf{X}_t \mid \mathbf{X}_{t-1})$ can be internally factored and include dependencies both between the time slices $t-1$ and t and within the slice t. Figure 3.2 shows a concrete example of a dynamic Bayesian network.

Given the specification of the distribution $P(\mathbf{X}_t \mid \mathbf{X}_{t-1})$ and an initial distribution $P(\mathbf{X}_0)$, a dynamic Bayesian network can be "unrolled" onto multiple time slices. This unrolled model corresponds to a classical Bayesian network.

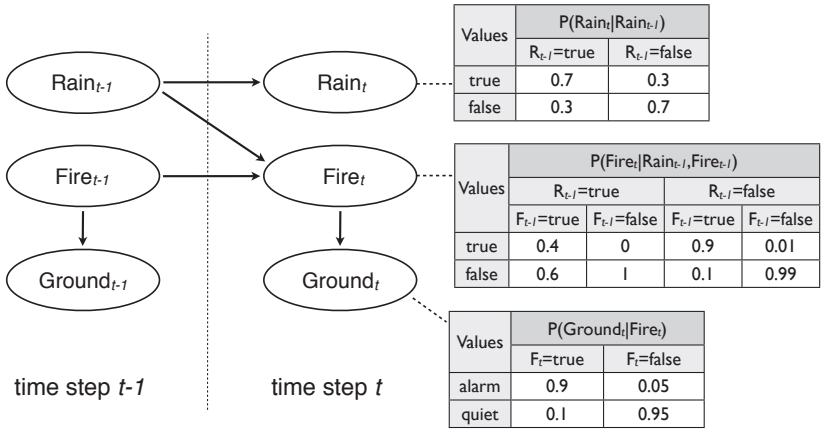


Figure 3.2: Example of a dynamic Bayesian network.

[4]The formalisation presupposes a discrete representation of time, with t representing the current time step.

Decision problems

Dynamic Bayesian networks are well-suited to represent temporal processes. However, in sequential decision tasks such as dialogue management, tracking the current state is only the first step of the reasoning process. The agent must also be able to calculate the relative utilities of the various actions that can be executed at that state. The second extension of Bayesian networks pertains therefore to the inclusion of actions and utilities in addition to state variables.

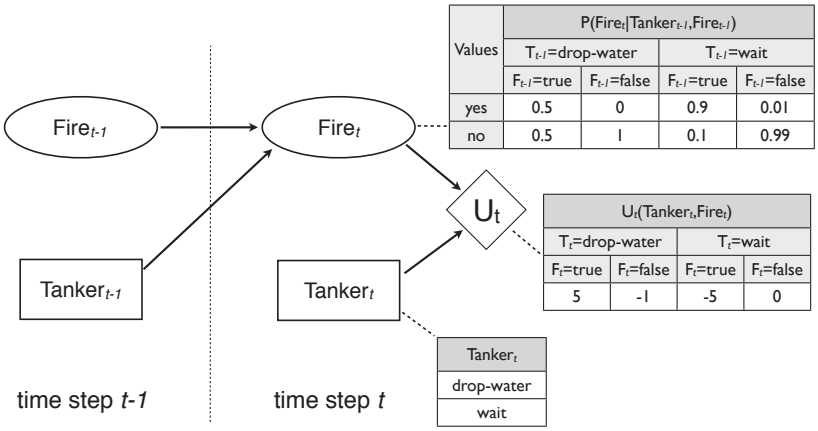
Decision networks[5] are Bayesian networks augmented with a representation of action variables and their corresponding utilities. Decision networks may include three classes of nodes:

1. *Chance nodes* correspond to the classical random variables described so far. Chance nodes are associated with conditional probability distributions that define the relative probabilities of the node values given the values in the parent nodes.

2. *Decision nodes* (sometimes also called action nodes) correspond to variables that are under the control of the system. The values of these nodes reflect an active choice made by the system to execute particular actions.

3. *Utility nodes* express the utilities (from the system's point of view) associated with particular situations expressed in the node parents. Typically, these parents combine both chance and decision variables. Utility nodes are coupled with utility distributions that associate each combination of values in the node parents with a specific (negative or positive) utility.

Decision networks combined with temporal dependencies are called *dynamic decision networks*. Figure 3.3 illustrates a dynamic decision network with a decision variable containing two alternative actions assigned to distinct utility values depending on the occurrence of fire.



Values	P(Fire$_t$|Tanker$_{t-1}$,Fire$_{t-1}$)			
	T_{t-1}=drop-water		T_{t-1}=wait	
	F_{t-1}=true	F_{t-1}=false	F_{t-1}=true	F_{t-1}=false
yes	0.5	0	0.9	0.01
no	0.5	1	0.1	0.99

U_t(Tanker$_t$,Fire$_t$)			
T_t=drop-water		T_t=wait	
F_t=true	F_t=false	F_t=true	F_t=false
5	-1	-5	0

Figure 3.3: Example of a dynamic decision network. By convention, chance nodes are represented with circles, decision nodes with squares, and utility nodes with diamonds.

[5]Decision networks are also called *influence diagrams*.

3.1.2 Inference

Exact and approximate algorithms

The main purpose of probabilistic graphical models is to evaluate *queries* – that is, calculate a posterior distribution over a subset of variables, usually given some evidence. Assuming a graphical model defining the joint probability distribution of a set of variables $\mathbf{X}$, a probability query is a posterior distribution of the form $P(\mathbf{Q} \mid \mathbf{e})$, where $\mathbf{Q} \subset \mathbf{X}$ denotes the query variables and $\mathbf{e}$ a specific assignment of values for the evidence variables. If the evidence does not contain any assignment, the query is reduced to the calculation of a marginal distribution. Decision networks can also be used to answer utility queries of the form $U(\mathbf{Q} \mid \mathbf{e})$. In this case, the query variables often correspond to decision nodes whose utility is to be determined.

A wide range of inference algorithms has been developed to efficiently evaluate these probability and utility queries. These algorithms can be either exact or approximate.

Exact algorithms calculate the precise posterior distribution corresponding to the query through a sequence of manipulations on the probability distributions associated with the graphical model. One popular algorithm for exact inference is variable elimination (Zhang and Poole, 1996). Variable elimination relies on dynamic programming techniques to evaluate a query through a sequence of matrix operations (summation and pointwise product). These operations are defined on so-called "factors" that represent CPDs in a matrix format. Variable elimination can be generalised to handle utility queries using joint factors (Koller and Friedman, 2009). Other algorithms such as message passing on clique trees can also be used (Jensen et al., 1990).

Unfortunately, exact inference remains difficult to scale to large, densely interconnected graphical models, and approximate techniques are often unavoidable in many practical domains. Algorithms for approximate inference in graphical models include approaches such as loopy belief propagation (Murphy et al., 1999), variational methods (Jordan et al., 1999), and a wide array of sampling techniques (MacKay, 1998). Popular sampling techniques include various flavours of importance sampling (Fung and Chang, 1989; Cheng and Druzdzel, 2000) and Markov Chain Monte Carlo (MCMC) approaches such as Gibbs sampling (Pearl, 1987; Gamerman and Lopes, 2006).

The evaluation of probabilistic queries is a difficult computational task in both exact and approximate inference techniques. In fact, inference on unconstrained Bayesian Networks is known to be #P-hard, which is a complexity class that is strictly harder than NP-complete problems. This holds both for exact inference (Cooper, 1990), but also – perhaps more surprisingly – for approximate inference (Dagum and Luby, 1993).

The `OpenDial` toolkit we have developed for this thesis implements two inference algorithms: generalised variable elimination (Koller and Friedman, 2009, p. 1103) and a specific type of importance sampling method called likelihood weighting (Fung and Chang, 1989), which we outline below. These algorithms are used in the processing workflow of the dialogue manager to (1) update the dialogue state upon the reception of new observations and (2) select system actions on the basis of this updated dialogue state. The details of this workflow will be provided in Chapter 4.

Likelihood weighting

To make our discussion of inference algorithms for graphical models more concrete, we describe below a simple but efficient sampling method called *likelihood weighting* (Fung and Chang, 1989),

which we have used in most of the experiments conducted in this thesis.

The key principle behind all types of sampling algorithms is to estimate the posterior distribution expressed in the query by collecting a large quantity of samples (i.e. assignments of values to the random variables) drawn from the graphical model. Likelihood weighting proceeds by sampling the random variables in the graphical model one by one, in topological order (i.e. from parents to children).[6] For instance, sampling the network in Figure 3.1 will start with the variable $Weather$, then $Fire$ (based on the value drawn for the parent $Weather$), and finally $Ground$ and $Satellite$ (based on the value drawn for $Fire$). In order to account for the evidence $\mathbf{e}$, every sample is associated with a specific *weight* that expresses the likelihood of the evidence given the assignment for all the other variables. For graphical models that include utility variables, samples also record the total utility accumulated for the sampled values.

The pseudocode in Algorithm 1 outlines the sampling procedure. The notation $\mathbf{e}(X_i)$ refers to the value specified for the variable X_i in the assignment $\mathbf{e}$.

Algorithm 1 : GET-SAMPLE $(\mathcal{B}, \mathbf{e})$

Input: Bayesian/decision network $\mathcal{B}$ and evidence $\mathbf{e}$
Output: Full sample drawn from $\mathcal{B}$ together with a weight and utility

Let $X_1, \ldots, X_n$ be a topological ordering for the variables in $\mathcal{B}$
Initialise sample $\mathbf{x} \leftarrow \langle \mathbf{e} \rangle$
Initialise weight $w \leftarrow 1$ and utility $u \leftarrow 0$
for $i = 1, \ldots, n$ **do**
  if X_i is a chance variable and is included in the evidence **then**
    $w \leftarrow w \times P\left(X_i = \mathbf{e}(X_i) \mid \mathbf{x}\right)$
  else if X_i is a chance or decision variable **then**
    $x_i \leftarrow$ sample value drawn from $P(X_i \mid \mathbf{x})$
    $\mathbf{x} \leftarrow \mathbf{x} \cup \langle x_i \rangle$
  else if X_i is a utility variable **then**
    $u \leftarrow u + X_i(\mathbf{x})$
  end if
end for
return $\mathbf{x}, w, u$

A large number of samples can be accumulated in this manner. Once enough samples are collected (or the inference engine has run out of time), the resulting posterior distribution for the query variables $\mathbf{Q}$ is derived by normalising the weighted counts associated with each value of the query variables, as shown in Algorithm 2. Algorithm 3 extends the procedure to the calculation of utility distributions. In this case, the utility values correspond to a weighted average of the sampled utilities.

3.1.3 Learning

We have so far pushed aside the question of how the distributions in the graphical model are exactly estimated. Early approaches often relied on distributions elicited from human experts based on plausible associations they have observed. Although useful in domains where no data is available,

[6] A partial order on the nodes can always be found since the network is a directed acyclic graph.

Algorithm 2 :LIKELIHOOD-WEIGHTING $(\mathcal{B}, \mathbf{Q}, \mathbf{e}, N)$

Input: Bayesian/decision network $\mathcal{B}$
Input: Set of query variables $\mathbf{Q}$ and evidence $\mathbf{e}$
Input: Number N of samples to draw
Output: Approximate posterior distribution $P(\mathbf{Q} \mid \mathbf{e})$

Let $\mathbf{W}$ be vectors of weighted counts for each possible value of $\mathbf{Q}$, initialised with zeros
for $i = 1 \rightarrow N$ **do**
  $\mathbf{x}, w \leftarrow$ GET-SAMPLE$(\mathcal{B}, \mathbf{e})$
  $\mathbf{W}[\mathbf{x}(\mathbf{Q})] \leftarrow \mathbf{W}[\mathbf{x}(\mathbf{Q})] + w$
end for
Normalise the weighted counts in $\mathbf{W}$
return $\mathbf{W}$

Algorithm 3 : LIKELIHOOD-WEIGHTING-UTILITY $(\mathcal{B}, \mathbf{Q}, \mathbf{e}, N)$

Input: (see above)
Output: Approximate utility distribution $U(\mathbf{Q} \mid \mathbf{e})$

Let $\mathbf{W}, \mathbf{U}$ be vectors of weighted counts for each possible value of $\mathbf{Q}$, initialised with zeros
for $i = 1 \rightarrow N$ **do**
  $\mathbf{x}, w, u \leftarrow$ GET-SAMPLE$(\mathcal{B}, \mathbf{e})$
  $\mathbf{W}[\mathbf{x}(\mathbf{Q})] \leftarrow \mathbf{W}[\mathbf{x}(\mathbf{Q})] + w$
  $\mathbf{U}[\mathbf{x}(\mathbf{Q})] \leftarrow \mathbf{U}[\mathbf{x}(\mathbf{Q})] + w \times u$
end for
Average the weighted utility counts $\mathbf{U}(\mathbf{q}) \leftarrow \frac{\mathbf{U}(\mathbf{q})}{\mathbf{W}(\mathbf{q})}$ $\forall$ values $\mathbf{q}$ of $\mathbf{Q}$
return $\mathbf{U}$

hand-crafted models are unfortunately difficult to scale (only models with a limited number of parameters can be elicited in such a manner), and are vulnerable to human errors and inaccuracies. It is therefore often preferable to automatically estimate these distributions from real world data – in other words, via statistical estimation based on a collection of examples in a training set.

Two distinct types of learning tasks can be distinguished:

1. The most common task is *parameter estimation*. Parameter estimation assumes the general structure of the graphical model (i.e. the dependencies between variables) is known, but not the parameters of the individual CPDs. Most discrete and continuous distributions are indeed "parametrised" – that is, they depend on the specification of particular parameters that define the exact shape of the distribution. A categorical distribution on k values has for instance k parameters that assign the relative probability of each outcome. Similarly, a normal distribution $\mathcal{N}(\mu, \sigma^2)$ is governed by its two parameters μ and σ^2.

2. The second possible learning task is *structure learning*. In structure learning, the agent must learn both the structure (i.e. the directed edges) and the parameters of the graphical model, given only the list of variables and the training data. This task is significantly more complex than parameter estimation, as the agent must simultaneously find which variables influence one another and estimate their corresponding conditional dependencies.

We shall concentrate in this thesis on the parameter estimation problem, which is by far the most common type of learning problem in dialogue management. As shall be argued in the next chapters, the dependency relations between state variables can often be derived from prior knowledge of the dialogue domain.

Maximum likelihood estimation

The most straightforward estimation method is called maximum likelihood estimation (MLE). Maximum likelihood estimation searches for the parameter values that provide the best "fit" for the data set. In other words, the parameters are set to the values that maximise the likelihood of the observed data. Given a data set $\mathcal{D}$, a graphical model and a set of parameters $\boldsymbol{\theta}$ to estimate for this model, the objective of MLE is to find the values $\boldsymbol{\theta}^*$ that maximise the probability $P(\mathcal{D}\,;\,\boldsymbol{\theta})$, which is the likelihood of the data set $\mathcal{D}$ given the parameter values for $\boldsymbol{\theta}$. This likelihood is often written in logarithmic form to transform the product of probabilities into a summation:

$$\boldsymbol{\theta}^* = \operatorname*{argmax}_{\boldsymbol{\theta}} P(\mathcal{D}\,;\,\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \log P(\mathcal{D}\,;\,\boldsymbol{\theta}) \tag{3.5}$$

If the data samples cover the complete set of variables in the model, this likelihood can be neatly decomposed in a set of local likelihoods, one for each CPD, and the $\boldsymbol{\theta}^*$ values can be derived in closed-form. For a categorical distribution, the maximum likelihood estimates will simply correspond to the relative counts of occurrences in the training data.

The learning problem becomes more complex for partially observed domains in which the data samples contain hidden variables. For the Bayesian network in Figure 3.1, an example of a partially observed sample is $[\mathit{Weather} = \mathit{mild}, \mathit{Ground} = \mathit{alarm}, \mathit{Satellite} = \mathit{quiet}]$, where the occurrence of fire is not specified. In such cases, the likelihood function is no longer decomposable, which implies that the maximum likelihood estimate is not amenable to a closed-form solution. The only alternative is to resort to iterative optimisation methods such as gradient ascent (Binder et al., 1997) and Expectation Maximisation (Green, 1990).

The main drawback of maximum likelihood estimation is its vulnerability to overfitting when learning from small data sets. For instance, if we only had collected one single data point $\mathit{Weather} = \mathit{cold}$ for the example above, the maximum likelihood estimate for the probability distribution $P(\mathit{Weather})$ would be $\langle 1, 0, 0\rangle$. In other words, maximum likelihood estimation does not take into account any prior knowledge about the relative probability of particular parameter hypotheses, which often leads to unreasonable estimates for low frequency events.

Bayesian learning

An alternative to maximum likelihood estimation is *Bayesian learning*. The key idea of Bayesian approaches to parameter estimation is to view the model parameters as random variables themselves and to derive their posterior distributions after observing the data. Bayesian learning starts with an initial prior over the range of parameter values and gradually refines this distribution through probabilistic inference based on the observation of the samples in the training data. Each distribution $P(X_i \mid \mathit{parents}(X_i))$ with unknown parameters is therefore associated with a parent node $\theta_{X_i \mid \mathit{parents}(X_i)}$ that define its possible parameters.

Parameter distributions are typically continuous (since probabilities are continuous values), and

often multivariate. Intuitively, we can think of the variable $\theta_{X_i \mid parents(X_i)}$ as expressing a "distribution over possible distributions".

Based on this formalisation, parameter estimation can be elegantly reduced to a problem of probabilistic inference over the parameters. Given a prior $P(\boldsymbol{\theta})$ on the parameter values and a data set $\mathcal{D}$, the posterior distribution $P(\boldsymbol{\theta} \mid \mathcal{D})$ is given by Bayes' rule:

$$P(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{P(\mathcal{D} ; \boldsymbol{\theta})\, P(\boldsymbol{\theta})}{P(\mathcal{D})} \tag{3.6}$$

Equation (3.6) allows us to calculate the posterior distribution $P(\boldsymbol{\theta} \mid \mathcal{D})$ through the use of standard inference algorithms for graphical models.

It is often convenient to encode the distributions of the parameter variables as *conjugate priors* of their associated CPD distribution. In such case, the prior $P(\boldsymbol{\theta})$ and posterior $P(\boldsymbol{\theta} \mid \mathcal{D})$ after observing the data points $\mathcal{D}$ are ensured to remain within the same distribution family. In particular, if the distribution of interest is a categorical distribution, its parameter distribution can be encoded with a Dirichlet distribution, which is known as the conjugate prior of categorical and multinomial distributions. A Dirichlet distribution is a continuous, multivariate distribution of dimension k (with k being the size of the multinomial) that is itself parametrised with so-called concentration hyperparameters denoted $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_k]$. Additional details about the formal properties of Dirichlet distributions can be found in Appendix A.

Figure 3.4 illustrates this Bayesian approach to parameter estimation for the variable $Weather$. As the variable possesses three alternative values, the allowed values for the parameter $\boldsymbol{\theta}_{Weather}$ are three-dimensional vectors $[\theta_{\text{Weather}(1)}, \theta_{\text{Weather}(2)}, \theta_{\text{Weather}(3)}]$, with the standard constraints on probability values: $\theta_{\text{Weather}(i)} \geq 0$ for $i = \{1, 2, 3\}$ and $\theta_{\text{Weather}(1)} + \theta_{\text{Weather}(2)} + \theta_{\text{Weather}(3)} = 1$. As we can observe in the figure, these constraints effectively limit the range of possible values to a 2-dimensional simplex. The $\boldsymbol{\alpha}$ hyperparameters can be intuitively interpreted as "virtual counts" of the number of observations in each category. In Figure 3.4, we can see that the hyperparameters $\boldsymbol{\alpha} = [5, 10, 5]$ lead to higher probability densities for parameters around the peak $\langle 0.25, 0.5, 0.25 \rangle$. As the number of observations increases, the Dirichlet distribution will gradually concentrate on a particular region of the parameter space until convergence.

In the case of completely observed data, Bayesian learning over several parameters can be decomposed into independent estimation problems (one for each parameter variable):

$$P(\mathcal{D} ; \boldsymbol{\theta}) = \prod_{\theta_i \in \boldsymbol{\theta}} P(\mathcal{D} ; \theta_i) \tag{3.7}$$

As in the maximum likelihood estimation case, the learning task becomes more complicated when dealing with partially observed data, as the posterior distribution can no longer be represented as a product of independent posteriors over each parameter. In this setting, the full posterior is often too complex to be amenable to an analytic solution. Sampling techniques can be applied to offer reasonable approximations of this posterior. As we shall see in Chapters 5 and 6, the work presented in this thesis is grounded in these approximate Bayesian learning methods. Table 3.1 briefly summarises the parameter estimation methods discussed in this section.

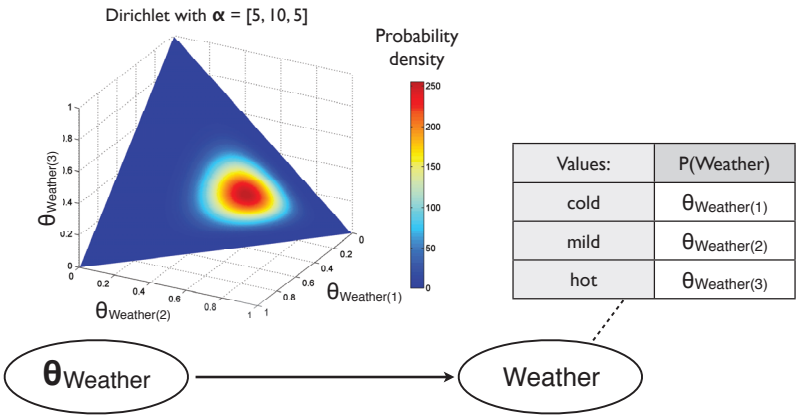


Figure 3.4: Bayesian network with variable $Weather$ and associated parameter $\theta_{Weather}$. As $Weather$ is a categorical distribution, $P(\theta_{Weather})$ is expressed as a Dirichlet distribution with three dimensions that reflect the relative probabilities for the three values in $Val(Weather)$.

Training data	**Maximum likelihood estimation**	**Bayesian learning**
Fully observed	Maximisation of local likelihood functions	Query on local posterior distribution over each parameter
Partially observed (hidden variables)	Iterative optimisation of global likelihood function	Query on full posterior over parameters via sampling

Table 3.1: Summary of parameter estimation approaches for directed graphical models.

3.2 Reinforcement learning

Dialogue management is fundamentally a problem of sequential decision-making under uncertainty. The objective of the dialogue manager is to select the "optimal" action – that is, the action that yields the maximum expected long-term utility for the agent. However, in many domains (including dialogue domains), the agent has no knowledge of the internal dynamics of the environment it finds itself in. It must therefore determine the best action to execute in any given state via a process of trial and error. Such a learning process is called *reinforcement learning* (RL). It is worth noting that reinforcement learning effectively strikes a middle ground between supervised and unsupervised learning. Contrary to supervised learning, the framework does not require the provision of "gold standard" examples. However, thanks to the rewards it receives from the environment, the agent is able to get a sense of the quality of its own decisions, an element which is absent in unsupervised learning methods.

We provide here a brief introduction to the central concepts in reinforcement learning, and

refer the interested reader to Sutton and Barto (1998) for more details.

3.2.1 Markov Decision Processes

Reinforcement learning tasks are typically based on the definition of a *Markov Decision Process* (MDP), which is a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ where:

- $\mathcal{S}$ is the state space of the domain and represents the set of all (mutually exclusive) states.
- $\mathcal{A}$ is the action space and represents the possible actions that can be executed by the agent.
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function and encodes the probability $P(s' \mid s, a)$ of reaching state s' after executing action a in state s.
- $R : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward function associated with the execution of action a in state s.

Markov Decision Processes can be explicitly represented as dynamic decision networks. As we can see from the graphical illustration in Figure 3.5, the state at time $t+1$ is dependent both on the previous state at time t and the action a_t performed by the system. After each action, the system receives a reward $r_t = R(s_t, a_t)$ that depends both on the state and selected action.

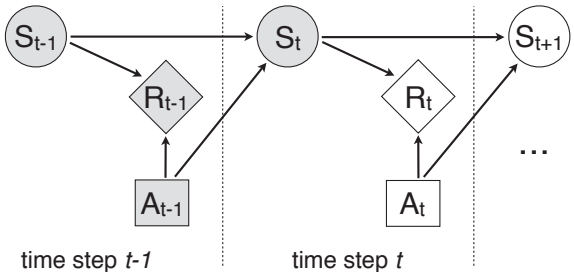


Figure 3.5: Graphical model of a Markov Decision Process (MDP) unfolded on a few time steps. Greyed entities indicate observed variables. In the MDP case, all past states, actions and rewards are observed, as well as the current state.

The objective of the learning agent is to extract an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ for the given MDP definition. This policy maps each state in $\mathcal{S}$ to the action that maximises the *expected return* of the agent for that state, which is the discounted sum of rewards, starting from the current state up to a potentially infinite horizon. In this sum, a geometric discount factor γ (with $0 \leq \gamma \leq 1$) indicates the relative worth of future rewards in regard to present ones. At one extreme, $\gamma = 0$ corresponds to a short-sighted agent only interested in its immediate reward, while at the other extreme, $\gamma = 1$ corresponds to an agent for which immediate and distant rewards are valued equally. For a given policy π, the expected return for an arbitrary state s in $\mathcal{S}$ is expressed through the *value function* $V^\pi(s)$:

$$V^\pi(s) = E\left\{ r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \;\middle|\; s_0 = s, \pi \text{ is followed} \right\} \tag{3.8}$$

$$= E\left\{ \sum_{i=0}^{\infty} \gamma^i r_i \;\middle|\; s_0 = s, \pi \text{ is followed} \right\} \tag{3.9}$$

where $r_i = R(s_i, \pi(s_i))$ is the reward received at time i after performing the action specified by the policy π in state s_i. Equation (3.9) can be rewritten in a recursive form, leading to the well-known Bellman equation (Bellman, 1957):

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) V^{\pi}(s') \tag{3.10}$$

In other words, the expected return in state s equals its immediate reward plus the expected return of its successor state. The recursive definition offered by the Bellman equation is crucial for many reinforcement learning methods, since it allows the value function to be estimated by an iterative process in which the value function is gradually refined until convergence.

Another useful concept is the action–value function $Q^{\pi}(s, a)$ that expresses the return expected after performing action a in state s and following the policy π afterwards:

$$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V^{\pi}(s') \tag{3.11}$$

The value and action–value functions are closely related, as $V^{\pi}(s) = Q^{\pi}(s, \pi(s))$.

The objective of the learning process is to find a policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that selects the action with highest expected return for all possible states: $V^{\pi^*}(s) \geq V^{\pi}(s)$ for any state s and policy π. For a given MDP, there is at least one policy that satisfies this constraint. The value and action–value functions for this optimal policy are respectively denoted V^* and Q^*.

Two families of reinforcement learning approaches can be used to determine this policy: *model-based* approaches and *model-free* approaches. As we shall see in Chapter 6, these optimisation techniques form the methodological basis for our own reinforcement learning approach to the estimation of dialogue management models.

Model-based approaches

Model-based approaches estimate an explicit model of the MDP and subsequently optimise a policy based on this model. The agent initially collects data by interacting with its environment and recording the reward and successor state following each action. Based on this data, standard parameter estimation methods (as outlined in Section 3.1.3) can be used to learn the MDP parameters, and the resulting model is applied to extract the corresponding optimal policy via dynamic programming (Bertsekas and Tsitsiklis, 1996) or Bayesian learning (Dearden et al., 1999).

The most well-known model-based algorithm is *value iteration*, which operates by estimating the value function via a sequence of updates based on Bellman's equation. Given a value function estimate V_k available at step k, the estimate at step $k + 1$ is calculated as:

$$V_{k+1}(s) \leftarrow \max_a R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a)\, V_k(s') \tag{3.12}$$

Once the iterations have converged, the optimal policy is straightforwardly derived as:

$$\pi^*(s) \leftarrow \operatorname*{argmax}_a \left(R(s, a) + \sum_{s' \in \mathcal{S}} P(s' \mid s, a)\, V^*(s') \right) \tag{3.13}$$

Model-free approaches

Model-free approaches skip the estimation of the underlying MDP model in order to directly learn Q^* functions from the agent's interaction experience. The main idea behind model-free methods is to let the agent try out different actions, observe their effects in terms of rewards and successor states, and use this information to refine the estimate of the Q^* function. The estimation process is, however, slightly more intricate than in model-based settings, since the Q-values are not directly accessible to the learning agent. The only feedback perceived by the agent are indeed the immediate rewards resulting from its actions and the subsequent observations.

Temporal-difference algorithms such as Q-learning (Watkins and Dayan, 1992) and SARSA (Rummery, 1995) are amongst the most popular model-free methods. These algorithms rely on Bellman's equation to incrementally improve the action–value estimates on the basis of the rewards resulting from the agent actions. Temporal-difference algorithms are also called "bootstrapping" methods as they approximate new Q-value estimates based on previous estimates.

The SARSA[7] algorithm has in particular been applied extensively for dialogue policy optimisation (Rieser and Lemon, 2011). The algorithm follows a simple learning procedure. Let s_t be a state at time t followed by a system action a_t. The execution of the system action a_t results in a reward r_{t+1} and a new state s_{t+1} which is itself followed by a second action a_{t+1}. On the basis of this sequence, the Q-value estimate for the first action a_t can be updated as:

$$Q_{k+1}(s_t, a_t) \leftarrow Q_k(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \, Q_k(s_{t+1}, a_{t+1}) - Q_k(s_t, a_t) \right] \tag{3.14}$$

where α represents the learning rate of the algorithm. The estimate $Q_{k+1}(s_t, a_t)$ is thus modified in direction of the value $[r_{t+1} + \gamma \, Q_k(s_{t+1}, a_{t+1}]$, with a learning step expressed by α. This operation is repeated for a large number of episodes until convergence.

One key question that must be addressed in both model-based and model-free approaches is how to efficiently explore the space of possible actions. The agent should indeed favour the selection of high-utility actions in most cases (since they are the ones of interest to the agent), but should also occasionally explore actions that are currently thought to be less effective to avoid being stuck in a suboptimal behaviour. This trade-off, called the *exploration–exploitation* dilemma, is one of the central research questions in reinforcement learning.

3.2.2 Partially Observable Markov Decision Processes

A limitation faced by MDP approaches is the assumption that the current state is fully observable. As we already noted in the previous chapter, this assumption does not hold for most dialogue systems, owing to the presence of multiple sources of uncertainty. As briefly alluded to in Chapter 2, an elegant solution to this shortcoming is to extend the MDP framework by allowing the state to be a hidden variable that is indirectly inferred from observations. Such an extension gives rise to a *Partially Observable Markov Decision Process* (POMDP). POMDPs are formally defined as tuples $\langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, Z \rangle$. As in a classical MDP, $\mathcal{S}$ represents the state space, $\mathcal{A}$ the action space, T the transition probability $P(s' \mid s, a)$ between states, and R the reward function $R(s, a)$. However, the actual state is no longer directly observable. Instead, the process is associated with an observation space $\mathcal{O}$ that expresses the set of possible observations that can be perceived by the system. The

[7] SARSA stands for "State-Action-Reward-State-Action", as a reference to the algorithm's processing sequence.

function Z defines the probability $P(o \mid s)$ of observing o in the current state s. Figure 3.6 provides a graphical illustration of the POMDP framework. As we can see, POMDPs can also be expressed as dynamic decision networks in which the state variable is not directly observed.

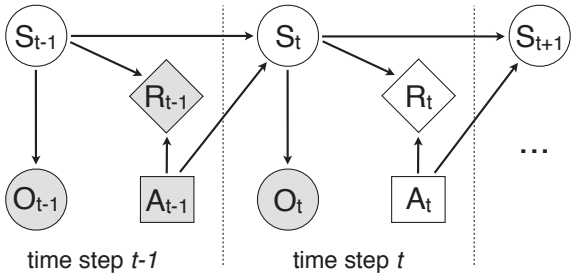


Figure 3.6: Graphical model of a Partially Observable Markov Decision Process (POMDP) unfolded on a few time steps. Compared to Figure 3.5, we notice that the state is no longer directly accessible but must be inferred from the observations and (predicted) state history.

The agent knowledge at a given time is represented by the *belief state* b, which is a probability distribution $P(s)$ over possible states. After each system action a and subsequent observation o, the belief state b is updated to b' in order to incorporate the new information. This belief update is a simple application of Bayes' theorem:

$$b'(s) = P(s' \mid b, a, o) = \frac{P(o \mid s')\ P(s' \mid b, a)}{P(o \mid b, a)} \tag{3.15}$$

$$= \eta\ P(o \mid s') \sum_{s} P(s' \mid s, a)\ b(s) \tag{3.16}$$

where $\eta = (P(o \mid b, a))^{-1}$ serves as a normalisation factor and is usually never calculated explicitly.

In the POMDP setting, a policy is a function $\pi = \mathcal{B} \rightarrow \mathcal{A}$ mapping each possible belief state to its optimal action. Mathematically, the belief state space $\mathcal{B}$ is a $(|\mathcal{S}|-1)$-dimensional simplex (where $|\mathcal{S}|$ is the size of the state space), which is a continuous and high-dimensional space. The optimisation of the dialogue policy is therefore considerably more complex than for MDPs. The value function V^{π} for a policy π is the fixed point of Bellman's equation:

$$V^{\pi}(b) = \sum_{s \in S} R(s, a) b(s) + \gamma \sum_{o \in \mathcal{O}} P(o \mid b, \pi(b))\ V^{\pi}(b') \tag{3.17}$$

where b' is the updated belief state following the execution of action $\pi(b)$ and the observation of o, as in Equation (3.16). The optimal value function $V^{*}(b)$ for finite-horizon problems is known to be piecewise linear and convex in the belief space – and can be approximated arbitrarily well in the case of infinite horizons – as proved by Sondik (1971). The value function can therefore be represented by a finite set of vectors, called α-vectors. Each vector α_i is associated with a specific action $a(i) \in \mathcal{A}$.[8] The vectors are of size $|\mathcal{S}|$ and $\alpha_i(s)$ is a scalar value representing the value of

[8] The converse is not true: each action can be associated with an arbitrary number of vectors.

action $a(i)$ in state s. Given these vectors, the value function simplifies to:

$$V^*(b) = \max_i \alpha_i \cdot b \tag{3.18}$$

And the policy π^* can be rewritten as:

$$\pi^*(b) = a\left(\operatorname*{argmax}_i(\alpha_i \cdot b)\right) \tag{3.19}$$

Extracting the α-vectors associated with a POMDP problem is, however, computationally challenging given the high-dimensional and continuous nature of the belief space, and exact solutions are intractable beyond toy domains. As proved by Papadimitriou and Tsitsiklis (1987), deriving the α-vectors for a given POMDP – also known as "solving" the POMDP – is a PSPACE-complete problem, which means that the best known algorithms will take time $2^{poly(n,h)}$ to solve a problem with n states and a planning horizon h..

A number of approximate solutions have been developed to address this limitation. One simple strategy is to rewrite the $Q(b, a)$ function in terms of the Q-values for the underlying MDP, as proposed by Littman et al. (1998):

$$Q(b, a) = \sum_{s \in \mathcal{S}} Q_{MDP}(s, a)\, b(s) \tag{3.20}$$

Although this approximation can work well in some settings, it essentially rests on the assumption that the state uncertainty will disappear after one action. It is therefore a poor model for information-gathering actions – that is, actions that do not change the actual state but might help in reducing state uncertainty. A typical example of such an action in the dialogue domain is a clarification request about the user intention.

Many approximations methods focus on reducing the complexity of the belief state space, notably through the use of grid-based approximations (Zhou and Hansen, 2001). The idea is to estimate the value function only at particular points within the belief simplex. At runtime, the action–value function for the current belief state b is then approximated to the value for the closest point in the grid according to some specific distance measure. Instead of a fixed grid, most recent solution methods rely on sampling techniques to perform local value updates (Pineau et al., 2003; Kurniawati et al., 2008; Shani et al., 2013). Belief compression has also proved to be useful (Roy et al., 2005). Yet another family of approximation methods directly search in the space of possible policies constrained to a particular form such as finite-state controllers (Hansen, 1998).

A final alternative, which we follow in Chapter 6 of this thesis, is to rely on online planning algorithms (Ross et al., 2008; Silver and Veness, 2010). The idea is to let the agent estimate the $Q(b, a)$ values at execution time via look-ahead search on a limited planning horizon. The planning horizon represents the number of future time steps considered by the planner. Compared to offline policies, the major advantage of online approaches is that the agent only needs to consider the current state to plan instead of enumerating all possible ones. It can also more easily adapt to changes in the reward or transition models, as the policy is not compiled in advance but generated at runtime. Moreover, online planning can also be used to simultaneously learn or refine these models during the interaction, as demonstrated by Ross et al. (2011). However, the available

planning time is more limited, since planning is in this case interleaved with system execution and must therefore satisfy real-time constraints.

Despite these recent advances, the optimisation of POMDP policies remains to a large extent an open research question in the fields of reinforcement learning and decision-theoretic planning. One important insight that transpires in much of the POMDP literature is the importance of exploiting the problem structure to reduce the complexity of the learning and planning problems (Pineau, 2004; Poupart, 2005). As detailed in the next chapter, the work presented in this thesis precisely attempts to transfer this insight into dialogue management.

3.2.3 Factored representations

In the previous pages, we modelled the system states and actions as atomic symbols. Such plain representations can unfortunately quickly lead to a combinatorial explosion of the state–action spaces. A more efficient alternative is to apply *factored* representations that decompose the state into distinct variables with possible conditional dependencies between one another. Action variables can also be decomposed in a similar manner. For an MDP, the state will take the form of a set of variables $\mathbf{S}_t$, and the transition function $P(\mathbf{S}_t \mid \mathbf{S}_{t-1}, \mathbf{A}_{t-1})$ be represented as a dynamic Bayesian network (Boutilier et al., 1999). The reward function can similarly be decomposed in a collection of utility variables $\mathbf{R}_t$ connected to relevant sets of state and action variables. In that case, the total utility is often defined as the addition of all utilities (Bacchus and Grove, 1995).

POMDPs can be factored in a similar way, with the inclusion of observation variables $\mathbf{O}_t$ connected to the state variables $\mathbf{S}_t$ through conditional dependencies $P(\mathbf{O}_t \mid \mathbf{S}_t)$. At time t, the observation variables $\mathbf{O}_t$ will be observed while the state variables $\mathbf{S}_t$ remain hidden. Figure 3.7 illustrates this factorisation. As we can observe from the figure, both plain and factored (PO)MDPs form specific cases of dynamic decision networks (cf. Section 3.1.1).

This factorisation of the domain variables through conditional independence assumptions leads to a more compact representation of the domain models and hence a substantial reduction in the number of associated parameters. In addition, the introduced structure can be directly exploited to e.g. improve the efficiency of probabilistic inference (Koller and Friedman, 2009) and the tractability of POMDP solution algorithms (Poupart, 2005).

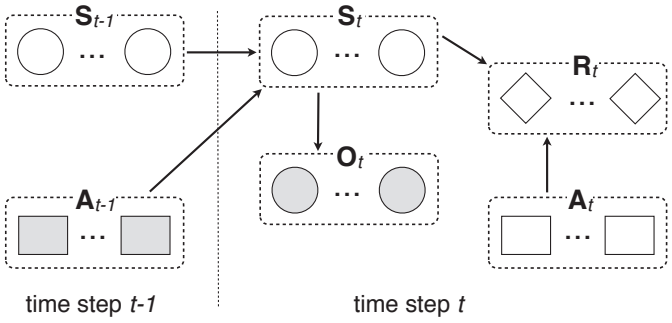


Figure 3.7: Factored representation of a POMDP with state variables $\mathbf{S}$, action variables $\mathbf{A}$, observation variables $\mathbf{O}$, and rewards $\mathbf{R}$.

3.3 Application to dialogue management

We have now reviewed the key ideas of probabilistic reasoning and reinforcement learning, and are ready to explain how these ideas can be practically transferred to the dialogue management task. After a brief review of supervised learning approaches, we detail how dialogue can be modelled as a Markov Decision Process or Partially Observable Markov Decision Process, and survey the various optimisation techniques that have been developed to automatically optimise dialogue policies for such models based on real or simulated interaction data.

3.3.1 Supervised approaches

The most straightforward use of statistical approaches to dialogue management is to learn dialogue policies in a supervised manner, based on external examples of dialogues. As discussed in the previous chapter, such examples are often collected via Wizard-of-Oz experiments where dialogue management is remotely performed by a human expert. The resulting training data is a sequence $\{\langle s_i, a_i \rangle : 1 \leq i \leq n\}$ of state–action pairs, where s_i is the state at time i and a_i the corresponding wizard action, which is assumed to reflect the best action to perform in this state. Most supervised learning approaches encode the dialogue state as a list of feature-value pairs, and the goal of the learning algorithm is to train a classifier $C : \mathcal{S} \rightarrow \mathcal{A}$ from states to actions that produces the best fit for this training data (modulo regularisation constraints), and will therefore "mimic" the decisions of the expert in similar situations. Various classifiers can be used for this purpose, such as maximum likelihood classification (Hurtado et al., 2005), decision trees (Lane et al., 2004), Naive Bayes (Williams and Young, 2003), and logistic regression (Rieser and Lemon, 2006; Passonneau et al., 2012).

The most important issue faced by supervised learning approaches is data sparsity, as only a fraction of the possible states can realistically be covered by the dialogue examples. Several generalisation techniques can be employed to alleviate this problem. The simplest is to encode the dialogue policy as a linear function of state features. The size of the feature set can be further reduced with feature selection (Passonneau et al., 2012). Yet another approach put forward by Hurtado et al. (2005) is to couple the classifier with a distance measure between states, thereby allowing the reuse of strategies learned from closely related states.

3.3.2 MDP dialogue policies

Instead of learning a dialogue policy by imitating the behaviour of human experts, the dialogue manager can also learn by itself the best action to perform in each possible conversational situation via repeated interactions with a (real or simulated) user. Dialogue management can be cast as a type of Markov Decision Process $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ in which:

- The state space $\mathcal{S}$ corresponds to the set of possible dialogue states, usually encoded as a list of feature-value pairs that capture relevant aspects of the current conversational context.

- The actions space $\mathcal{A}$ corresponds to the set of (verbal or non-verbal) actions that can be executed by the system.

- The transition function T captures the "dynamics" of the conversation, and indicates how the dialogue state is expected to change as a result of the system actions (and in particular how the user is expected to respond).

- The reward function R expresses the objectives and costs of the application. A common reward function is to assign a high positive value for the successful completion of the task, a high negative value for a failure, and a small negative value for soliciting the user to repeat or clarify their intention.

A common misunderstanding should be clarified at this point. The representation of dialogue as a Markov Decision Process implies that the transition function only considers the previous state and action to predict the next state. It has occasionally been argued that this formalisation makes the decision-making process oblivious to non-local aspects of the dialogue history. This argument, however, is essentially invalid, as the dialogue state is not limited to the mere representation of the last dialogue act but may express any features related to the interaction context, including variables recording dialogue histories of arbitrary length, long-term user intentions, and so forth. It is ultimately up to the system designer to decide which features are deemed relevant to describe the current state of the dialogue.

Dialogue state representation

As for supervised learning methods, MDP-based reinforcement learning approaches to dialogue management encode the dialogue state in terms of feature-value pairs. The dialogue state is thus factored into a number of independent variables (one for each feature). Most early approaches adopted crude state representations with features limited to the status of the slots to fill and the last user utterance (Levin et al., 2000; Singh et al., 2000; Scheffler and Young, 2002). The voice-enabled email client described in Walker (2000) captured additional measures related to the overall task progress, history of previous system attempts, confidence thresholds and timing information. There has also been some work on the automatic identification of relevant state variables, using methods from structure learning in decision networks (Paek and Chickering, 2006) and feature selection (Tetreault and Litman, 2006).

Henderson et al. (2008) were (to our knowledge) the first to explore the extension of reinforcement learning methods to large state spaces based on rich representations of the conversational context. Inspired by information state approaches to dialogue management, their state space captures detailed information such as the complete history of dialogue acts and fine-grained representations of the task status, amounting to a total of 10^{386} possible states. Such rich state representations allow the dialogue manager to exploit much broader contextual knowledge in its decision-making. However, it also creates important challenges regarding action selection, as generalisation techniques are necessary to scale up the learning procedure to such large state spaces.

Policy optimisation (and associated generalisation techniques)

For most dialogue domains, the reward is fixed in advance by the system designer and reflects the task objectives. It can also correspond to a performance metric such as PARADISE (Walker, 2000). In such a case, the performance metric is represented as a linear combination of quantitative measures whose weight is empirically estimated via multivariate linear regression from surveys of

user satisfaction. The quantitative measures of dialogue performance can include measures of *task success* (e.g. ratio of completed vs. failed tasks, κ agreement for slot-filling applications), *dialogue quality* measures (e.g. number of repetitions, interruptions, ASR rejection prompts) and *dialogue efficiency* measures (e.g. number of utterances per dialogue, total elapsed time).

The transition probabilities, however, are typically unknown. Although a few methods have been developed to estimate an explicit transition model for the dialogue domain based on observed interactions (Singh et al., 2002; Tetreault and Litman, 2006), the bulk of the research on dialogue policy optimisation has so far relied on model-free techniques. Due to the significant amounts of data necessary to reach convergence, it is often impossible to directly learn the value function from interactions with real users for most practical domains. A user simulator is instead used to provide unlimited supplies of interactions to the reinforcement learning algorithm. Several options are available to construct this user simulator. The first option is to design the simulator manually based on specific assumptions about the user behaviour (Pietquin and Dutoit, 2006; Schatzmann et al., 2007a). The simulator can also be derived in a data-driven manner from existing corpora (Georgila et al., 2006). In this case, user simulation can be interpreted as a way to expand the initial data set. The third available option is to exploit Wizard-of-Oz studies to tune the simulator parameters (Rieser and Lemon, 2010b). In addition to user modelling, the simulator should also integrate error modelling techniques in order to simulate the errors that may appear along the speech recognition and understanding pipeline (Schatzmann et al., 2007b; Thomson et al., 2012)

The key benefit of user simulation lies in the possibility to explore a large number of possible dialogue trajectories. Such kind of simulation is of great use for prototyping dialogue policies and experimenting with alternative setups (and in particular with various levels of noise and errors). However, simulated interactions are not as valuable as real interactions and offer no guarantee of matching the behaviour and error patterns of actual users. Paek (2006) also argues against the practice of training and evaluating the accuracy of dialogue policies on the same simulator.

A key problem in reinforcement learning methods is the *curse of dimensionality*: The number of parameters grows exponentially with the size of the state and action spaces (Sutton and Barto, 1998). In order to scale the learning procedure to larger domains, factorisation and generalisation techniques are often necessary. An early example of this line of research is the work of Paek and Chickering (2006) on the use of graphical models for dialogue policy optimisation. The domain employed in their experiments was a speech-enabled web browser. Their strategy was to explicitly represent dialogue management as a dynamic decision network and learn both the structure and parameters of this network from user simulations. The state space included features extracted from the interaction logs. Based on the simulated dialogues, the learning algorithm could automatically discover the subset of state variables that were relevant for decision-making as well as the transition probabilities between these variables. They also experimented with various Markov orders (first- versus second-order Markov models) to analyse the impact of longer state histories on the system performance. After learning the decision network, dynamic programming techniques were used to extract a dialogue policy that is optimal with respect to the learned models.

Henderson et al. (2008) demonstrate how to reduce the complexity of model-free learning techniques by way of function approximation. As the large size of their state–action space prevented the use of classical tabular representations for the $Q(s, a)$ function, they instead relied on function

approximation to define the Q-values as a linear function of state features:

$$Q(s, a) = f(s)^T w_a \tag{3.21}$$

where $f(s)$ is a feature vector extracted from the state and w_a a weight vector for action a. The weight vectors were learned with the SARSA algorithm based on a fixed corpus (thereby avoiding the use of user simulators). To further refine the estimation of Q-values, their approach combined estimates from both supervised and reinforcement learning in their final model.

Hierarchical abstraction is another way to reduce the search space of the optimisation process, as shown by Cuayáhuitl (2011). Instead of viewing action selection as a single monolithic policy, the selection is decomposed in their work into multiple levels, each responsible for a specific decision problem. This decomposition is formally expressed via a hierarchical extension of MDPs called Semi-Markov Decision Processes. Contrary to classical Markov Processes, Semi-Markov Decision Processes allow for actions that take a variable amount of time to complete their execution (a characteristic called *temporal abstraction*). Each level in the hierarchy is associated with its own subset of state and action variables. This modular approach allows a complex policy to be split into a sequence of simpler decisions. Such hierarchical formalisation is particularly natural for task-oriented dialogues, which are known to exhibit rich attentional and intentional structures, as notably argued in the seminal work of Grosz and Sidner (1986). It also bears similarities with the approach presented by Litman and Allen (1987) on dialogue understanding based on a plan structure.

Finally, it is worth noting that several researchers have attempted to combine the benefits of supervised and reinforcement learning methods by initialising an RL algorithm with a policy estimated via supervised methods (Williams and Young, 2003; Rieser and Lemon, 2006).

3.3.3 POMDP dialogue policies

To capture the uncertainty associated with some state variables, dialogue can be explicitly modelled as a Partially Observable Markov Decision Process $\langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, Z \rangle$. Modelling a dialogue domain as a POMDP is similar in most respects to the MDP formalisation. The observations typically correspond to the possible N-best lists that can be generated by the speech recogniser and NLU modules, and can also include observations perceived via other modalities.

Dialogue state representation

POMDP approaches express state uncertainty through the definition of a belief state b, which is a probability distribution $P(s)$ over possible states. After a system action a in belief state b followed by observation o, the belief state b is updated according to Equation (3.16), repeated here for convenience:

$$b' = P(s' \mid b, a, o) = \eta \, P(o \mid s') \sum_s P(s' \mid s, a) \, b(s) \tag{3.16}$$

Belief update requires the specification of two probabilistic models: the observation model $P(o \mid s')$ and the transition model $P(s' \mid s, a)$. Many POMDP approaches to dialogue management factor the state s into (at least) three distinct variables $s = \langle a_u, i_u, c \rangle$, where a_u is the last user

dialogue act, i_u the current user intention(s), and c the interaction context.[9] Assuming that the observation o only depends on the last user act a_u, and that a_u depends on both the user intention i_u and the last system action a_m, Equation (3.16) is then rewritten as:

$$b' = P(a'_u, i'_u, c' \mid b, a_m, o) \tag{3.22}$$

$$= \eta\, P(o \mid a'_u)\, P(a'_u \mid i'_u, a_m) \sum_{i_u, c'} P(i'_u \mid i_u, a_m, c') P(c')\, b(i_u) \tag{3.23}$$

The transition model is decomposed in this factorisation into distinct distributions:

1. The distribution $P(a'_u \mid i'_u, a_m)$ is called the *user action model* and defines the probability of a particular user action given the underlying intention and the last system act. It expresses the likelihood of the user action a'_u following the system action a_m and the compatibility of a'_u with the user intention i'_u (Young et al., 2010).

2. The distribution $P(i'_u \mid i_u, a_m, c')$ is the *user goal model* and captures how the user intention is likely to change as a result of the context and system actions.

These two distributions are usually derived from collected interaction data. A graphical illustration of this state factorisation is shown in Figure 3.8.

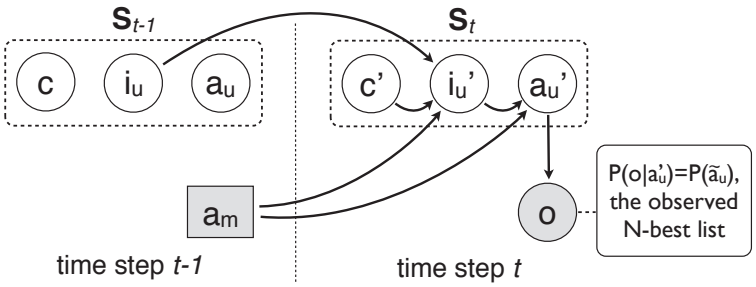


Figure 3.8: Common factorisation of the state space for a POMDP-based dialogue system, where c represents the dialogue context, i_u the user intention, a_u the last user dialogue act, a_m the last system act, and o the observation. The representation omits the conditional dependencies for the variable c', as this variable is by nature contingent on the dialogue domain.

The observation model $P(o \mid a'_u)$ is often rewritten as $P(\tilde{a}_u)$, the dialogue act probability in the N-best list provided by the speech recognition and semantic parsing modules (cf. Section 2.2.2), based on the following approximation:

$$P(o \mid a'_u) = \frac{P(a'_u \mid o)\, P(o)}{P(a'_u)} \approx P(a'_u \mid o) = P(\tilde{a}_u) \tag{3.24}$$

The above approximation rests on the assumption of uniform distributions for the possible observations in the absence of further evidence. Since the probabilities $P(\tilde{a}_u)$ are provided at runtime

[9] Some approaches also define a specific variable for the dialogue history (Young et al., 2010).

by the ASR and NLU modules, the approximation does not require an explicit specification of the observation set $\mathcal{O}$ and model Z at design time. This modelling choice has the advantage of circumventing the statistical estimation of the observation model, a difficult problem since the number of possible N-best lists is theoretically infinite. A few approaches have nevertheless been developed to estimate explicit observation models for small dialogue domains. Williams et al. (2008) investigated in particular how to integrate observations that include continuous-valued ASR confidence scores into a classical POMDP framework and devised ad-hoc density functions for this purpose. Chinaei et al. (2012) showed how an observation model could be empirically estimated from interaction data with a bag-of-words approach.

In the seminal work of Roy et al. (2000) that first introduced the POMDP framework to dialogue management, the state is represented by a single variable expressing the user intention, and hand-crafted models were used for the belief update. Zhang et al. (2001) extended the aforementioned approach by introducing a factored state representation based on Bayesian networks, where the state includes both the user intention and the system state. Williams et al. (2005) and Young et al. (2010) further refined this factorisation by decomposing the dialogue state into three distinct variables that respectively represent the last user dialogue act, the user intention and the dialogue history. The related work of Thomson and Young (2010) used Bayesian networks to encode fine-grained dependencies between the various slots expressed in the user intention. Bui et al. (2009) augmented this representation with a specific variable for the user's affective state. Substantial work has also been devoted to the inclusion of non-verbal observations and environmental factors into the dialogue state. In the human–robot interaction domain, Prodanov and Drygajlo (2003) and Hong et al. (2007) have applied Bayesian networks for inferring the underlying user intention based on observations arising from both verbal and non-verbal sources.

Policy optimisation

POMDP solution methods can in theory be applied to extract the dialogue policy from any given model specification, as shown by Williams and Young (2007) and Williams et al. (2008). Such a strategy is, however, only suitable for relatively small action-state spaces and requires the specification of an explicit observation model to extract the α-vectors corresponding to the optimal policy. Most recent POMDP approaches have instead focused on the use of reinforcement learning to derive a dialogue policy from interactions with a user simulator (Young et al., 2010; Thomson and Young, 2010; Daubigney et al., 2012b).

The need for effective generalisation techniques is heightened in reinforcement learning for POMDPs, as dialogue policies defined in partially observable environments must be defined in a high-dimensional, continuous belief state space. Approximating the $Q^*(b, a)$ function is thus crucial. One useful approximation method is to reduce the full belief state to a simpler representation such as the "summary state" described by Williams and Young (2005). In addition, the estimation of the action–value function can be further simplified by relying on techniques such as grid-based discretisations (Young et al., 2010) and linear function approximation (Thomson and Young, 2010; Daubigney et al., 2012b). Finally, non-parametric methods based on Gaussian processes have recently been proposed (Gašić et al., 2013).

Reinforcement learning is considerably more difficult for POMDPs than for MDPs due to the partial observability of the state. Png and Pineau (2011) showed how model-based reinforcement

learning can be cast in a Bayesian framework. As described in more detail in Section 6.1, the key idea of Bayesian reinforcement learning is to maintain an explicit probability distribution over the POMDP parameters. This distribution is then gradually refined as more data is observed through Bayesian inference. Our own work on reinforcement learning of rule parameters also follows that line of work, as shall be explained in Chapter 6.

Most POMDP-based approaches to dialogue management assume that the reward model can be encoded in advance by the system designer, with a few notable exceptions. Atrash and Pineau (2009) describe a Bayesian approach to estimate a reward model based on gold-standard actions provided by an oracle. Boularias et al. (2010) and Chinaei and Chaib-draa (2012) present alternative approaches based on inverse reinforcement learning (IRL) for POMDPs. Their main idea was to exploit Wizard-of-Oz data for a voice-enabled intelligent wheelchair to automatically infer a reward model. This task of inferring a reward model from expert demonstrations is a prototypical instance of *inverse reinforcement learning*: The agent observes how an expert performs the task and must find the hidden reward model that best explains this behaviour. Inverse reinforcement learning in partially observable domains is, however, difficult to scale beyond small domains due to the complexity of the optimisation problem (Choi and Kim, 2011).

3.4 Summary

We have presented in this chapter the foundations of probabilistic modelling applied to dialogue, and have surveyed a range of theoretical concepts related to graphical models, reinforcement learning in both fully and partially observable domains, as well as the practical exploitation of these techniques to dialogue management.

The first section of this chapter focused on the use of efficient representations of probability and utility models. We described how directed graphical models could capture various probability and utility distributions. We reviewed the main properties of Bayesian networks, dynamic Bayesian networks, and dynamic decision networks, and described the most important algorithms for inference and parameter estimation that have been tailored for these graphical models. The theoretical appeal of graphical models for the representation of complex stochastic phenomena is elegantly summarised by Jordan (1998, p. 1):

> "Graphical models, a marriage between probability theory and graph theory, provide a natural tool for dealing with two problems that occur throughout applied mathematics and engineering – uncertainty and complexity. In particular, they play an increasingly important role in the design and analysis of machine learning algorithms. Fundamental to the idea of a graphical model is the notion of modularity: a complex system is built by combining simpler parts. Probability theory serves as the glue whereby the parts are combined, ensuring that the system as a whole is consistent and providing ways to interface models to data. Graph theory provides both an intuitively appealing interface by which humans can model highly interacting sets of variables and a data structure that lends itself naturally to the design of efficient general-purpose algorithms."

The second section of this chapter laid down the core concepts and techniques in the field of reinforcement learning. We described the notion of a Markov Decision Process (MDP), composed

of a set of states, a set of actions, a transition function describing temporal relations between states, and a reward function encoding the utilities of particular actions. We explained how MDPs can be extended to capture partial observability (leading to POMDPs), and how policies can be optimised for both MDPs and POMDPs using model-based and model-free techniques.

The final section translated these formal representations and optimisation techniques to the practical problem of dialogue management. Three learning strategies were distinguished: supervised learning, reinforcement learning with MDPs, and reinforcement learning with POMDPs. We surveyed a variety of approaches that differ along dimensions such as the representation of the dialogue state, the learning algorithm employed for the optimisation, the type and structure of the models that are to be estimated, and the source of data samples that is employed for this estimation. We discussed the respective merits and limitations of these dialogue optimisation strategies, and stressed in particular the importance of factorisation and generalisation methods to handle the complexity of real-world dialogue domains. The next chapter will now present in detail our own approach to this prominent problem.