Chapter 1

Expectation-Maximization methods for solving (PO)MDPs and optimal control problems

Marc Toussaint[1], *Amos Storkey*[2] *and Stefan Harmeling*[3]

As this book demonstrates, the development of efficient probabilistic inference techniques has made considerable progress in recent years, in particular with respect to exploiting the structure (e.g., factored, hierarchical or relational) of discrete and continuous problem domains. In this chapter we show that these techniques can be used also for solving Markov Decision Processes (MDPs) or partial observable MDPs (POMDPs) when formulated in terms of a structured dynamic Bayesian network (DBN).

The problems of planning in stochastic environments and inference in state space models are closely related, in particular in view of the challenges both of them face: scaling to large state spaces spanned by multiple state variables, or realizing planning (or inference) in continuous or mixed continuous-discrete state spaces. Both fields developed techniques to address these problems. For instance, in the field of planning, they include work on Factored Markov Decision Processes (Boutilier et al., 1995; Koller and Parr, 1999; Guestrin et al., 2003; Kveton and Hauskrecht, 2005), abstractions (Hauskrecht et al., 1998), and relational models of the environment (Zettlemoyer et al., 2005). On the other hand, recent advances in inference techniques show how structure can be exploited both for exact inference as well as making efficient approximations. Examples are message passing algorithms (loopy Belief Propagation, Expectation Propagation), variational approaches, approximate belief representations (particles, Assumed Density Filtering, Boyen-Koller) and arithmetic compilation (see, e.g., Minka, 2001; Murphy, 2002; Chavira et al., 2006).

In view of these similarities one may ask whether existing techniques for probabilistic inference can directly be translated to solving stochastic planning problems. From a complexity theoretic point of view, the equivalence between inference and planning is well-known (see, e.g., Littman et al., 2001). Inference methods have been applied before to optimal decision making in Influence Diagrams (Cooper, 1988; Pearl, 1988; Shachter, 1988). However, contrary to MDPs, these methods focus on a finite number of decisions and a non-stationary policy, where optimal decisions are found by recursing backward starting from the last decision (see (Boutilier et al., 1999) and (Toussaint, 2009) for a discussion of MDPs versus Influence Diagrams). More recently, Bui et al. (2002) have used inference on Abstract Hidden Markov Models for policy recognition, i.e., for reasoning about executed behaviors, but do not address the problem of computing optimal policies from such inference. Attias

[1]TU Berlin
[2]U Edinburgh
[3]MPI Tübingen

(2003) proposed a framework which suggests a straight-forward way to translate the problem of planning to a problem of inference: A Markovian state-action model is assumed, which is conditioned on a start state s_0 and a goal state s_T. Here, however, the total time T has to be fixed *ad hoc* and the MAP action sequence that is proposed as a solution is not optimal in the sense of maximizing an expected future reward. Raiko and Tornio (2005) introduced the same idea independently in the context of continuous state stochastic control and called this optimistic inference control. Verma and Rao (2006) used inference to compute plans (considering the maximal probable explanation (MPE) instead of the MAP action sequence) but again the total time has to be fixed and the plan is not optimal in the expected return sense.

We provide a framework that translates the problem of maximizing the discounted expected future return in the infinite-horizon MDP (or general DBN) into a problem of likelihood maximization in a related mixture of finite-time MDPs. This allows us to use expectation maximization (EM) for computing optimal policies, utilizing arbitrary inference techniques in the E-step. We can show that this optimizes the discounted expected future return for arbitrary reward functions and without assuming an ad hoc finite total time. The approach is generally applicable on any DBN-description of the problem whenever we have efficient inference techniques for this structure. DBNs allow us to consider structured representations of the environment (the world state) as well as the agent (or multiple agents, in fact).

The next section introduces our Likelihood Maximization approach for solving Markov Decision Processes. This will involve the introduction of a mixture of variable length DBNs for which we can show equivalence between likelihood maximization and maximization of expected future return. Section 1.2 in detail derives an EM algorithm. Here, the key are efficient inference algorithms that handle the mixture of variable length processes. The derived algorithms are applicable on arbitrary structured DBNs. In section 1.3 we reconsider the basic MDP case, explain the relation of the EM algorithm to Policy and Value Iteration, and demonstrate the approach using exact inference on a discrete maze and Gaussian belief state propagation in non-linear stochastic optimal control problems.

In section 1.4 we consider a non-trivial DBN representation of a POMDP problem. We propose a certain model of an agent (similar to finite state controllers, FSCs) that uses an internal memory variable as a sufficient representation of history which gates a reactive policy. We use this to learn sufficient memory representations (e.g., counting aisles) and primitive reactive behaviors (e.g., aisle or wall following) in some partially observable maze problems. This can be seen in analogy to the classical Machine Learning paradigm of learning latent variable models of data for bootstrapping interesting internal representations (e.g., ICA) — but here generalized to the learning of latent variable models of successful behavior. Section 1.5 will conclude this paper and discuss existing follow-up work and future directions of research.

1.1 Markov Decision Processes and likelihood maximization

A Markov Decision Process (MDP, (Kaelbling et al., 1996)) is a stochastic process on the random variables of state s_t, action a_t, and reward r_t, as defined by the

initial state distribution	$P(s_0 = s)$,
transition probability	$P(s_{t+1} = s' \mid a_t = a, s_t = s)$,
reward probability	$P(r_t = r \mid a_t = a, s_t = s)$,

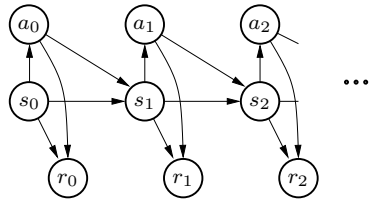

Figure 1.1: Dynamic Bayesian network for a MDP. The x states denote the state variables, a the actions and r the rewards.

policy $$P(a_t = a \,|\, s_t = s; \pi) =: \pi_{as} \ .$$

We assume the process to be stationary (none of these quantities explicitly depends on time) and call the expectation $\mathcal{R}(a,s) = \mathrm{E}\{r \,|\, a, s\} = \sum_r r \ P(r \,|\, a, s)$ the reward function. In model-based Reinforcement Learning the transition and reward probabilities are estimated from experience (see, e.g., (Atkeson and Santamaría, 1997)). In section 1.5.1 we discuss follow-up work that extends our framework to the model-free case. The random variables s_t and a_t can be discrete or continuous whereas the reward r_t is a real number. Figure 1.1 displays the dynamic Bayesian network for an infinite-horizon Markov Decision Process (MDP).

The free parameter of this DBN is the policy π with numbers $\pi_{as} \in [0,1]$ normalized w.r.t. a. The problem we address is *solving the MDP*:

Definition 1.1.1. Solving an MDP means to find a parameter π of the infinite-horizon DBN in Figure 1.1 that maximizes the expected future return $V^\pi = \mathrm{E}\{\sum_{t=0}^\infty \gamma^t \ r_t; \pi\}$, where $\gamma \in [0,1)$ is a discount factor.

The classical approach to solving MDPs is anchored in Bellman's equation, which simply reflects the recursive property of the future discounted return

$$\sum_{t=0}^\infty \gamma^t \ r_t = r_0 + \gamma \ \Big[\sum_{t=0}^\infty \gamma^t \ r_{t+1} \Big] \tag{1.1}$$

and consequently of its expectation conditioned on the current state,

$$V^\pi(s) = \mathrm{E}\{ \sum_{t=0}^\infty \gamma^t \ r_t \ \Big| \ s_0 = s; \pi \} = \sum_{s',a} P(s'|a,s) \ \pi_{as} \ [\mathcal{R}(a,s) + \gamma \ V^\pi(s')] \ . \tag{1.2}$$

Standard algorithms for computing value functions can be viewed as iterative schemes that converge towards the Bellman equation or as directly solving this linear equation w.r.t. V by matrix inversion.

In contrast, our general approach is to translate the problem of solving an MDP into a problem of likelihood maximization. There are different approaches for such a translation. One issue to be considered is that the quantity we want to maximize (the expected future return) is a *sum* of expectations in every time slice, whereas the likelihood in Markovian models is the *product* of observation likelihoods in each time slice. A first idea for achieving equivalence is to introduce exponentiated rewards as observation likelihoods – but that turns out non-equivalent (see Remark (iii) in the appendix section .1).

Toussaint and Storkey (2006); Toussaint et al. (2006) introduced an alternative based on a mixture of finite-length processes. Intuitively, the key argument for this

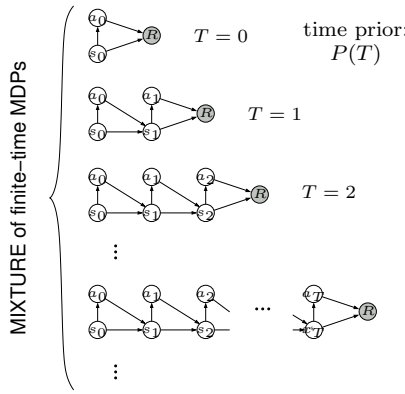

Figure 1.2: Mixture of finite-time MDPs.

approach is perhaps the question *Where do I start the backward sweep?* In all EM approaches we need to compute a posterior over trajectories in the E-step so that we can update the policy in the M-step. In finite-length Markov processes, such inference can efficiently be done by a forward-backward sweep (Baum-Welch). If the process is infinite it is unclear where in the future to start (anchor) the backward sweep. However, when introducing a binary reward event there is a very intuitive solution to this: Let us simply declare that the reward event occurs at some time T in the future, without knowing what T is, and we start the backward sweep from that time T backward. In that way we can start computing backward and forward messages in parallel without having to estimate some horizon ad hoc, and decide when to stop if there is sufficient overlap between the forward and backward propagated messages (Toussaint and Storkey, 2006). When we choose a geometric prior $P(T) = (1-\gamma)\gamma^t$ this turns out to implement the discounting correctly.

The mixture model is in some respects different to the original MDP but the likelihood of "observing reward" in this mixture model is proportional to the expected future return in the original MDP. The reasons for this choice of approach are related to inference (performing a backward pass) without pre-fixing a finite time horizon T, the handling of discounting rewards, and also to the resulting relations to standard Policy and Value Iteration, as it will later become more clear.

We define the mixture model as follows. Let $\xi = (s_{0:T}, a_{0:T})$ denote a state-action sequence of length T, and let R be a random event (binary variable) with $P(R \,|\, a, s)$ proportional to the reward function $\mathcal{R}(a, s)$, for instance as in (Cooper, 1988),

$$P(R \,|\, a, s) = \frac{\mathcal{R}(a, s) - \min(\mathcal{R})}{\max(\mathcal{R}) - \min(\mathcal{R})} \ . \tag{1.3}$$

Each finite-time MDP defines the joint

$$\begin{aligned} &P(R, \xi \,|\, T; \pi) \\ &= P(R \,|\, a_T, s_T)\ P(a_T \,|\, s_T; \pi)\ \Big[\prod_{t=0}^{T-1} P(s_{t+1} \,|\, a_t, s_t)\ P(a_t \,|\, s_t; \pi)\Big]\ P(s_0) \ . \end{aligned} \tag{1.4}$$

That is, each finite-time MDP has the same initialization, transition, and reward probabilities as the original MDP but (i) it ends at a finite time T and (ii) it emits a single binary reward R only at the finial time step.

Now let T be a random variable with prior $P(T)$. The mixture of finite-time MDPs is given by the joint

$$P(R, \xi, T; \pi) = P(R, \xi \,|\, T; \pi)\ P(T)\ . \tag{1.5}$$

Note that this defines a distribution over the random variable ξ in the space of variable length trajectories. Figure 1.2 illustrates the mixture. We find

Theorem 1.1.1. *When introducing binary rewards R such that $P(R \,|\, a, s) \propto \mathcal{R}(a, s)$ and choosing the geometric time prior $P(T) = \gamma^T(1-\gamma)$, maximizing the likelihood*

$$L(\pi) = P(R; \pi) \tag{1.6}$$

of observing reward in the mixture of finite-time MDPs is equivalent to solving the original MDP.

Given the way we defined the mixture, the proof is straight-forward and illustrated in Figure 1.3.

Proof. Let H be some horizon for which we later take the limit to ∞. We can rewrite the value function of the original MDP as

$$V^\pi = \sum_{a_{0:H}, s_{0:H}} \Big[P(s_0)\ \pi(a_0 \,|\, s_0)\ \prod_{t=1}^{H} \pi(a_t \,|\, s_t)\ P(s_t \,|\, a_{t\text{-}1}, s_{t\text{-}1})\Big] \Big[\sum_{T=0}^{H} \gamma^T \mathcal{R}(a_T, s_T)\Big] \tag{1.7}$$

$$= \sum_{T=0}^{H} \gamma^T \sum_{a_{0:H}, s_{0:H}} \mathcal{R}(a_T, s_T) P(s_0)\ \pi(a_0 \,|\, s_0)\ \prod_{t=1}^{H} \pi(a_t \,|\, s_t)\ P(s_t \,|\, a_{t\text{-}1}, s_{t\text{-}1}) \tag{1.8}$$

$$= \sum_{T=0}^{H} \gamma^T \sum_{a_{0:T}, s_{0:T}} \mathcal{R}(a_T, s_T) P(s_0)\ \pi(a_0 \,|\, s_0)\ \prod_{t=1}^{T} \pi(a_t \,|\, s_t)\ P(s_t \,|\, a_{t\text{-}1}, s_{t\text{-}1}) \tag{1.9}$$

$$= \sum_{T=0}^{H} \gamma^T\ \mathrm{E}_{a_{0:T}, s_{0:T} \,|\, \pi}\{\mathcal{R}(a_T, s_T)\}\ . \tag{1.10}$$

In the second line we pulled the summation over T to the front. Note that the second and third line are really different: the product is taken to the limit T instead of H since we eliminated the variables $a_{T+1:H}, s_{T+1:H}$ with the summation. The last expression has already the form of a mixture model, where T is the mixture variable, γ^T is the mixture weight ($P(T) = \gamma^T(1-\gamma)$ the normalized geometric prior), and the last term is the expected reward in the final time slice of a *finite-time* MDP of length T (since the expectation is taken over $a_{0:T}, s_{0:T} \,|\, \pi$).

The likelihood in our mixture model can be written as

$$L(\pi) = P(R; \pi) \tag{1.11}$$

$$= (1-\gamma) \sum_{T=0}^{\infty} \gamma^T P(R \,|\, T; \pi) \tag{1.12}$$

$$\propto (1-\gamma)\ \mathrm{E}_{a_{0:T}, s_{0:T} \,|\, \pi}\{\mathcal{R}(a_T, s_T)\} \tag{1.13}$$

$$= (1-\gamma)\ V^\pi\ . \tag{1.14}$$

The proportionality stems from the definition $P(R \,|\, a_t, s_t) \propto \mathcal{R}(a_T, s_T)$ of R. $\square$

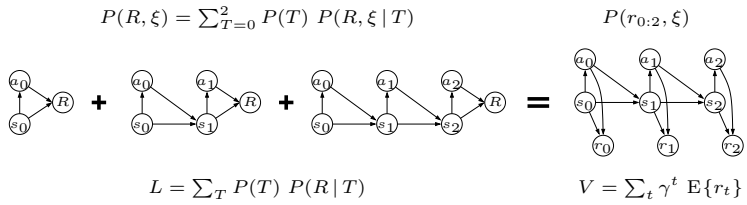

Figure 1.3: Illustration of the equivalence between the mixture and the original MDP.

In appendix .1 we remark on the following points in some more detail:

(i) the interpretation of the mixture with death probabilities of the agent,

(ii) the difference between the models w.r.t. the correlation between rewards,

(iii) and approaches to consider exponentiated rewards as observation likelihoods.

1.2 Expectation Maximization in mixtures of variable length Dynamic Bayesian Networks

The algorithms we will derive in this section are independent from the context of MDPs. We investigate the general case of a variable length stationary Markov process where we have observations only at the start and the final time. The length of the process is unknown and we assume a mixture of variable length processes where the length prior is the geometric distribution $P(T) = \gamma^T(1-\gamma)$ for $\gamma \in [0,1)$. We first derive EM algorithms for an unstructured Markov process on one random variable which is then easily generalized to structured DBNs.

1.2.1 Single variable case

Let ξ be a random variable in the domain $\mathcal{X}^* = \bigcup_{T=0}^{\infty} \mathcal{X}^{T+1}$, that is, a variable length trajectory in $\mathcal{X}$ (in other terms, a finite but arbitrary length string over the alphabet $\mathcal{X}$). We use the notation $\xi = (x_0, .., x_T)$ for a length-T trajectory and write $|\xi| = T$. We consider T a random variable and assume an auxiliary binary random variable R depending on the final state x_T which represents some generic observation. Specifically, we consider the joint

$$\begin{aligned} P(R,\xi,T;\theta) &= P(R\,|\,\xi;\theta)\ P(\xi\,|\,T;\theta)\ P(T) \\ &= P(R\,|\,x_T;\theta)\ \Big[\prod_{t=0}^{T\text{-}1} P(x_{t+1}\,|\,x_t;\theta)\Big]\ P(x_0;\theta)\ \delta_{|\xi|T}\ P(T)\ . \end{aligned} \tag{1.15}$$

Here, $\delta_{|\xi|T}$ is one for $|\xi| = T$ and zero otherwise; $P(T)$ is a prior over the length of the process; $P(x_0;\theta)$ is the start distribution of the process; $P(x_{t+1}\,|\,x_t;\theta)$ is the transition probability; and $P(R\,|\,x_T;\theta)$ is the probability of the R-event which depends only on the final state. The joint is normalized since, when summing over all $\xi \in \mathcal{X}^*$ with all lengths $L = |\xi|$,

$$\begin{aligned} \sum_{\xi\in\mathcal{X}^*} P(\xi\,|\,T;\theta) &= \sum_{L=0}^{\infty} \sum_{x_0,..,x_L} \Big[\prod_{t=0}^{L\text{-}1} P(x_{t+1}\,|\,x_t;\theta)\Big]\ P(x_0;\theta)\ \delta_{LT} \\ &= \sum_{x_0,..,x_T} \Big[\prod_{t=0}^{T\text{-}1} P(x_{t+1}\,|\,x_t;\theta)\Big]\ P(x_0;\theta) = 1\ . \end{aligned} \tag{1.16}$$

For Expectation Maximization (EM) we assume R is observed and we want to find parameters θ of the joint that maximize the likelihood $P(R;\theta) = \sum_{T,\xi} P(R,\xi,T;\theta)$. The length T of the process and the whole trajectory ξ itself are latent (non-observed) variables. Let $q(\xi,T)$ be a distribution over the latent variables. Consider

$$F(\theta,q) := \log P(R;\theta) - D\big(q(\xi,T) \,\big\|\, P(\xi,T \,|\, R;\theta)\big) \tag{1.17}$$

$$= \log P(R;\theta) - \sum_{\xi,T} q(\xi,T) \log \frac{q(\xi,T)}{P(\xi,T \,|\, R;\theta)} \tag{1.18}$$

$$= \sum_{\xi,T} q(\xi,T) \log P(R;\theta) + \sum_{\xi,T} q(\xi,T) \log P(\xi,T \,|\, R;\theta) + H(q) \tag{1.19}$$

$$= \sum_{\xi,T} q(\xi,T) \log P(R,\xi,T;\theta) + H(q) \tag{1.20}$$

EM will start with an initial guess of θ, then iterate finding a q that maximizes F for fixed θ in the form of (1.17), and then finding a new θ that maximizes F for fixed q in the form of (1.20). Let us first address the M-step. This will clarify which quantities we actually need to compute in the E-step.

The M-step computes $\operatorname{argmax}_\theta$ of

$$F(\theta,q) = \sum_{\xi,T} q(\xi,T) \log P(R,\xi,T;\theta) + H(q) \tag{1.21}$$

$$= \sum_{\xi,T} q(\xi,T) \Big[\log P(R \,|\, x_T;\theta) + \log P(\xi \,|\, T;\theta) + \log P(T) \Big] + H(q) \tag{1.22}$$

$$= \sum_{\xi,T} q(\xi,T) \Big[\log P(R \,|\, x_T;\theta) + \log P(\xi \,|\, T;\theta) \Big] \quad + \text{terms indep of } \theta \tag{1.23}$$

$$= \sum_{\xi} \sum_{T=0}^{\infty} q(\xi,T) \Big[\log P(R \,|\, x_T;\theta) + \sum_{t=0}^{T\text{-}1} \log P(x_{t+1} \,|\, x_t;\theta) \Big] \tag{1.24}$$

$$\begin{aligned} &= \sum_{\xi} \sum_{T=0}^{\infty} q(\xi,T) \log P(R \,|\, x_T;\theta) \\ &\quad + \sum_{\xi} \sum_{t=0}^{\infty} \sum_{T=t+1}^{\infty} q(\xi,T) \log P(x_{t+1} \,|\, x_t;\theta) \end{aligned} \tag{1.25}$$

$$\begin{aligned} &= \sum_{x} \Big[\sum_{T=0}^{\infty} q(x_T = x, T) \Big] \log P(R \,|\, x_T = x;\theta) \\ &\quad + \sum_{x',x} \Big[\sum_{t=0}^{\infty} \sum_{T=t+1}^{\infty} q(x_{t+1} = x', x_t = x, T) \Big] \log P(x_{t+1} = x' \,|\, x_t = x;\theta) \end{aligned} \tag{1.26}$$

The last line uses that the process and the reward are stationary (i.e., $P(x_{t+1} = x' \,|\, x_t = x;\theta)$ does not explicitly depend on t and $P(R \,|\, x_T = x;\theta)$ does not depend explicitly on T). The last equation clarifies that the E-step actually only needs to return the quantities in the brackets. The exact E-step is

$$q^*(\xi,T) = P(\xi,T \,|\, R;\theta^{\text{old}}) = \frac{P(R \,|\, \xi,T;\theta^{\text{old}})\; P(\xi \,|\, T;\theta^{\text{old}})\; P(T)}{P(R;\theta^{\text{old}})} \tag{1.27}$$

Let us investigate the bracket terms in (1.26) for the exact E-step in more detail:

$$\sum_{T=0}^{\infty} q^*(x_T, T) = \frac{1}{P(R;\theta^{\text{old}})}\ P(R \,|\, x_T;\theta^{\text{old}})\ \sum_{T=0}^{\infty} P(x_T;\theta^{\text{old}})\ P(T) \tag{1.28}$$

$$\begin{aligned}\sum_{t=0}^{\infty}\sum_{T=t+1}^{\infty} q^*(x_{t+1}, x_t, T) = \frac{1}{P(R;\theta)}\ \sum_{t=0}^{\infty}\sum_{T=t+1}^{\infty} P(R \,|\, x_{t+1}, T;\theta^{\text{old}}) \\ \cdot P(x_{t+1} \,|\, x_t;\theta^{\text{old}})\ P(x_t;\theta^{\text{old}})\ P(T)\end{aligned} \tag{1.29}$$

At this point we use the property of the geometric length prior

$$P(T{=}t+\tau) = \frac{1}{1-\gamma}\ P(T{=}t)\ P(T=\tau) \tag{1.30}$$

$$\begin{aligned}\sum_{t=0}^{\infty}\sum_{T=t+1}^{\infty} q^*(x_{t+1}, x_t, T) = \frac{1}{P(R;\theta)(1-\gamma)}\ \sum_{t=0}^{\infty}\sum_{\tau=1}^{\infty} P(R \,|\, x_{t+1}, T{=}t+\tau;\theta^{\text{old}}) \\ \cdot P(T{=}\tau)\ P(x_{t+1} \,|\, x_t;\theta^{\text{old}})\ P(x_t;\theta^{\text{old}})\ P(T{=}t)\end{aligned} \tag{1.31}$$

$$\begin{aligned}= \frac{1}{P(R;\theta)(1-\gamma)}\ \Big[\sum_{\tau=1}^{\infty} P(R \,|\, x_{t+1}, T{=}t+\tau;\theta^{\text{old}})\ P(T{=}\tau)\Big] \\ \cdot P(x_{t+1} \,|\, x_t;\theta^{\text{old}})\ \Big[\sum_{t=0}^{\infty} P(x_t;\theta^{\text{old}})\ P(T{=}t)\Big]\end{aligned} \tag{1.32}$$

In the last line we used that $P(R \,|\, x_{t+1}, T{=}t+\tau;\theta^{\text{old}})$ does not explicitly depend on t but only on the time-to-go τ. We finally define quantities

$$\alpha(x_t) := \sum_{t=0}^{\infty} P(x_t;\theta^{\text{old}})\ P(T{=}t) \tag{1.33}$$

$$\beta(x_{t+1}) := \frac{1}{1-\gamma}\ \sum_{\tau=1}^{\infty} P(R \,|\, x_{t+1}, T{=}t+\tau;\theta^{\text{old}})\ P(T{=}\tau) \tag{1.34}$$

$$= \frac{1}{1-\gamma}\ \sum_{\tau=0}^{\infty} P(R \,|\, x'_t, T{=}t+\tau;\theta^{\text{old}})\ P(T{=}\tau+1) \tag{1.35}$$

such that the relevant parts of $F(\theta, q^*)$ for the M-step can be written more compactly. Equation (1.26) now reads

$$\begin{aligned}F(\theta, q^*) = \sum_x \Big[P(R \,|\, x_T;\theta^{\text{old}})\ \alpha(x)\Big] \log P(R \,|\, x_T;\theta) \\ + \sum_{x',x} \Big[\beta(x')\ P(x' \,|\, x;\theta^{\text{old}})\ \alpha(x)\Big] \log P(x_{t+1} \,|\, x_t;\theta)\end{aligned} \tag{1.36}$$

Note, $\alpha(x)$ and $\beta(x)$ are just quantities that are defined in equations (1.33) and (1.35) and useful for the EM algorithm, but have no explicit interpretation yet. They are analogous to the typical forward and backward messages in Baum-Welch, but different in that there are no observations and that they incorporate the whole mixture over variable length processes and exploit the geometric length prior. If one likes, α can be interpreted as the last-state-occupancy-probability averaged over the mixture of all length processes; and β can be interpreted as (is proportional to) the probability of observing R in the future (not immediately) averaged over the mixture

of all length processes. The explicit M-step depends on the kind of parameterization of $P(R \,|\, x_T; \theta)$ and $P(x_{t+1} \,|\, x_t; \theta)$ but is straight-forward to derive from (1.36). We will derive explicit M-steps in the (PO)MDP context in the next section.

1.2.2 Explicit E-step algorithms

In the remainder of this section we describe efficient algorithms to compute $\alpha(x)$ and $\beta(x)$ as defined in (1.33) and (1.35). For brevity we write $\boldsymbol{P} \equiv P(x' \,|\, x; \theta^{\text{old}})$ as a matrix and $\boldsymbol{\alpha} \equiv \alpha(x)$ and $\boldsymbol{S} \equiv P(x_0 = x)$ as a vectors. The $\boldsymbol{\alpha}$ quantity can be computed iteratively in two ways: We define another quantity $\boldsymbol{a}_t$ (which directly corresponds to the typical forward message) and from (1.33) get

$$\boldsymbol{a}_t := P(x_t = x) = \boldsymbol{P}\ \boldsymbol{a}_{t\text{-}1}\ , \quad \boldsymbol{a}_0 = \boldsymbol{S} \tag{1.37}$$

$$\boldsymbol{\alpha}_h = \sum_{t=0}^{h} \boldsymbol{a}_t\ P(T = t) = \boldsymbol{\alpha}_{h\text{-}1} + (1-\gamma)\gamma^h\ \boldsymbol{a}_h\ , \quad \boldsymbol{\alpha}_0 = (1-\gamma)\ \boldsymbol{S} \tag{1.38}$$

Iterating both equations together can be used to compute $\boldsymbol{\alpha}$ approximately as $\boldsymbol{\alpha}_h$ in the limit $h \to \infty$. Alternatively we can use

$$\begin{aligned}
\boldsymbol{\alpha}_h &= \sum_{t=0}^{h} \boldsymbol{a}_t\ P(T = t) = (1-\gamma)\ \Big[\boldsymbol{a}_0 + \sum_{t=1}^{h} \boldsymbol{a}_t\ \gamma^t\Big] \\
&= (1-\gamma)\ \Big[\boldsymbol{S} + \sum_{t=1}^{h} \boldsymbol{P}\ \boldsymbol{a}_{t\text{-}1}\ \gamma^t\Big] = (1-\gamma)\ \Big[\boldsymbol{S} + \boldsymbol{P}\ \sum_{t=0}^{h\text{-}1} \boldsymbol{a}_t\ \gamma^{t+1}\Big] \\
&= (1-\gamma)\ \boldsymbol{S} + \gamma\ \boldsymbol{P}\ \boldsymbol{\alpha}_{h\text{-}1}
\end{aligned} \tag{1.39}$$

as a direct recursive equation for $\boldsymbol{\alpha}_h$. Analogously we have two ways to compute $\boldsymbol{\beta}$ (a row vector). With $\boldsymbol{R} \equiv P(R \,|\, x_T = x)$ the direct computation of (1.35) is:

$$\boldsymbol{b}_\tau := P(R \,|\, x_t = x, T = t + \tau) = \boldsymbol{b}_{\tau\text{-}1}\ \boldsymbol{P}\ , \quad \boldsymbol{b}_0 = \boldsymbol{R} \tag{1.40}$$

$$\boldsymbol{\beta}_h = \frac{1}{1-\gamma}\ \sum_{\tau=0}^{h} \boldsymbol{b}_\tau\ P(T = \tau + 1) = \boldsymbol{\beta}_{h\text{-}1} + \gamma^{h+1}\ \boldsymbol{b}_h(x)\ , \quad \boldsymbol{\beta}_0 = \gamma\ \boldsymbol{b}_0 \tag{1.41}$$

And a second way to compute $\boldsymbol{\beta}$ is

$$\begin{aligned}
\boldsymbol{\beta}_h &= \frac{1}{1-\gamma}\ \sum_{\tau=0}^{h} \boldsymbol{b}_\tau\ P(T = \tau + 1) = \gamma \boldsymbol{b}_0 + \sum_{\tau=1}^{h} \boldsymbol{b}_\tau\ \gamma^{\tau+1} \\
&= \gamma \boldsymbol{R} + \sum_{\tau=1}^{h} \boldsymbol{b}_{\tau\text{-}1}\ \boldsymbol{P}\ \gamma^{\tau+1} = \gamma \boldsymbol{R} + \gamma\ \Big[\sum_{\tau=0}^{h\text{-}1} \boldsymbol{b}_\tau\ \gamma^{\tau+1}\Big]\ \boldsymbol{P} \\
&= \gamma \boldsymbol{R} + \gamma\ \boldsymbol{\beta}_{h\text{-}1}\ \boldsymbol{P}
\end{aligned} \tag{1.42}$$

in the limit $h \to \infty$. Note that, in the context of MDPs, this equation is exactly equivalent to *Policy Evaluation*, i.e., the computation of the value function for a fixed policy. We will discuss this in more detail in section 1.3.2.

When choosing to compute also the $\boldsymbol{a}_t$ and $\boldsymbol{b}_t$ quantities we can use them to compute the length posterior and likelihood,

$$P(R \,|\, T = t + \tau) = \sum_x P(R \,|\, x_t = x, T = t + \tau)\ P(x_t = x) = \boldsymbol{b}_\tau^\top\ \boldsymbol{a}_t \tag{1.43}$$

$$P(R) = \sum_T P(R \,|\, T)\ P(T) \tag{1.44}$$

$$P(T \,|\, R) = P(R \,|\, T)\ P(T)/P(R) \tag{1.45}$$

$$\mathrm{E}\{T \,|\, R\} = \sum_T T\ P(T \,|\, R) \tag{1.46}$$

In particular, equations (1.45) and (1.44) can be computed while iterating (1.37) and (1.40) and thereby provide a heuristic to choose the horizon (stopping criterion of the iteration) on the fly. On the other hand (1.39) and (1.42) are update equations for α and β which can be used in an incremental E-step: we can reuse the α and β of the previous EM-iterations as an initialization and iterate equations (1.39) and (1.42) only a few steps. This corresponds to computing parts of α and β with old parameters θ^{old} from previous EM-iterations and only the most recent updates with the current parameters. The horizon h implicitly increases in each EM-iteration. Algorithms 1 and 2 explicitly describe the standard E-step and the incremental version.

Algorithm 1 Standard E-step

Input: vectors $\boldsymbol{S}$, $\boldsymbol{R}$, matrix $\boldsymbol{P}$, scalars γ, H
Output: vectors $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $P(T|R)$, scalars $P(R)$, $E\{T|R\}$

initialize $\boldsymbol{a} = \boldsymbol{S}$, $\boldsymbol{b} = \boldsymbol{R}$, $\boldsymbol{\alpha} = \boldsymbol{a}$, $\boldsymbol{\beta} = \gamma\ \boldsymbol{b}$, $L(0) = \boldsymbol{a}^\top \boldsymbol{b}$
for $h = 1$ **to** H **do**
  $\boldsymbol{a} \leftarrow \boldsymbol{P}\ \boldsymbol{a}$
  $L(2h-1) = \gamma^{2h-1}\ \boldsymbol{a}^\top \boldsymbol{b}$
  $\boldsymbol{b} \leftarrow \boldsymbol{b}\ \boldsymbol{P}$
  $L(2h) = \gamma^{2h}\ \boldsymbol{a}^\top \boldsymbol{b}$
  $\boldsymbol{\alpha}$ += $\gamma^h\ \boldsymbol{a}$
  $\boldsymbol{\beta}$ += $\gamma^{h+1}\ \boldsymbol{b}$
end for
$\boldsymbol{L}$ *= $1 - \gamma$
$\boldsymbol{\alpha}$ *= $1 - \gamma$
$P(R) = \sum_{t=0}^{2H} L(t)$
$P(T{=}t|R) = L(t)/P(R)$
$E\{T|R\} = [\sum_{t=0}^{2H} tL(t)]/P(R)$

Algorithm 2 Incremental E-step

Input: vectors $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{S}$, $\boldsymbol{R}$, matrix $\boldsymbol{P}$, scalars γ, H
Output: vector $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, scalar $P(R)$

for $h = 1$ **to** H **do**
  $\boldsymbol{\alpha} \leftarrow (1-\gamma)\ \boldsymbol{S} + \gamma\ \boldsymbol{P}\ \boldsymbol{\alpha}$
  $\boldsymbol{\beta} \leftarrow \gamma\ [\boldsymbol{R} + \boldsymbol{\beta}\ \boldsymbol{P}]$
end for
$P(R) = \boldsymbol{a}^\top \boldsymbol{R}$

1.2.3 Structured DBN case

In the case of a structured DBN the process is defined on more than one variable. Generally the transition probability $P(x_{t+1} \,|\, x_t; \theta)$ is then replaced by a set of factors

(or conditional probability tables) that describes the coupling between two consecutive time slices. It is straight-forward to generalize algorithms 1 and 2 to exploit the structure in such a DBN: In each time slice we now have several random variables $s_t^1, .., s_t^k, a_t^1, .., a_t^l$. We choose the notation $s_t^1, .., s_t^k$ for a set of variables which are a separator of the Markov process and $a_t^1, .., a_t^l$ are the remaining variables in a time slice. For exact inference we have to maintain quantities $\alpha(s)$ and $\beta(s)$ over the separator clique. To exploit the DBN structure we need to replace the transition matrix multiplications (lines 3 and 5 in Algorithm 1, and lines 2 and 3 in Algorithm 2) with other inference techniques. The simplest solution is to use the elimination algorithm. For instance, instead of the matrix multiplication $a \leftarrow Pa$ (line 3 of Algorithm 1), we think of P as a list of factors over the variables (s_t, a_t, s_{t+1}) that couple two time slices, a is a factor over the "left" separator clique (s_t), we pass the list of factors $\{P, a\}$ to the elimination algorithm and query for the marginal over the "right" separator clique (s_{t+1}). This yields the new assignment to a. When we choose a good elimination order this procedure is equivalent to the Junction Tree method described in (Murphy, 2002).

Concerning the M-step, the energy expression (1.36) generalizes to

$$\begin{aligned} F(\theta, q^*) = &\sum_{a,s} \Big[P(R \,|\, a, s; \theta^{\text{old}})\ P(a \,|\, s; \theta^{\text{old}})\ \alpha(s) \Big] \log P(R \,|\, a, s; \theta)\ P(a \,|\, s; \theta) \\ &+ \sum_{s',a,s} \Big[\beta(s')\ P(s' \,|\, a, s; \theta^{\text{old}})\ P(a \,|\, s; \theta^{\text{old}})\ \alpha(s) \Big] \log P(s' \,|\, a, s; \theta)\ P(a \,|\, s; \theta) \end{aligned} \tag{1.47}$$

1.3 Application to MDPs

1.3.1 Expectation-Maximization with a tabular policy

A standard MDP is a DBN with random variables s_t and a_t in each time slice where the state s_t is a separator. In the simplest case we parameterize the policy $P(a_t \,|\, s_t; \theta)$ using a full CPT,

$$P(a_t = a \,|\, s_t = s; \theta) = \pi_{as}\ . \tag{1.48}$$

The energy (1.47) reads (neglecting terms independent of π)

$$\begin{aligned} F(\theta, q^*) = &\sum_{a,s} \Big[P(R|a, s)\ \pi_{as}^{\text{old}}\ \alpha(s) \Big] \log \pi_{as} \\ &+ \sum_{s',a,s} \Big[\beta(s')\ P(s'|a, s)\ \pi_{as}^{\text{old}}\ \alpha(s) \Big] \log \pi_{as}\ . \end{aligned} \tag{1.49}$$

Since π_{as} is constrained to normalize over a for each s this energy is maximized by

$$\pi_{as}^{\text{new}} = \pi_{as}^{\text{old}} \Big[P(R|a, s) + \sum_{s'} \beta(s')\ P(s'|a, s) \Big]\ . \tag{1.50}$$

The two terms in the brackets correspond to the expected immediate reward plus the expected future reward as predicted by β – the brackets are exactly equivalent to the classical Q-function.

In the case of a plain unstructured MDP we can also derive a greedy and usually faster version of the M-step, given as

$$\forall_s :\ \pi_{as}^{\text{new}} = \delta(a, a^*(s))\ , \quad a^*(s) = \operatorname*{argmax}_a \Big[P(R|a, s) + \sum_{s'} \beta(s')\ P(s'|a, s) \Big]\ . \tag{1.51}$$

This update corresponds to a greedy version of the previous M-step. If we would iterate (1.50) without recomputing the bracket term each time (skipping intermediate E-steps) we would converge to this greedy M-step. Further, as we know from Reinforcement Learning, the greedy M-step can be thought of as exploiting our knowledge that the optimal policy must be a deterministic one (in the fully observable case).

Note however that this does not generalize to arbitrarily structured DBNs. In fact, in the POMDP case that we investigate later we are not aware of such a greedy version of the M-step.

1.3.2 Relation to Policy Iteration and Value Iteration

So far we have developed our approach for a plain unstructured MDP. It turns out that in this case the E- and M-step are very closely related to the policy evaluation and update steps in standard Policy Iteration.

We introduced the mixture of MDPs in a way such that the likelihood is proportional to the expected future return. Hence, for the unstructured MDP, $b(s) := P(R \mid s_t = s, T = t+\tau; \pi)$ as defined in equation (1.40) is proportional to the expected reward in τ time steps in the future conditioned on the current state. This is also the value function of the finite-time MDP of length τ. Hence, the $\beta(s)$ defined in (1.35) is proportional to the value function $V^\pi(s)$ of the original MDP. Analogously, the bracket term in the M-step (1.50) is proportional to the Q-function $Q^\pi(a, s)$ in the original MDP.

We conclude that the E-step in an unstructured MDP is a form of Policy Evaluation since it also yields the value function. However, quite different to traditional Policy Evaluation, the E-step also computes α's, i.e., probabilities to visit states given the current policy, which may be compared to previous approaches like Diverse Densities (in the context of subgoal analysis (McGovern and Barto, 2001)) or Policy Search by Density Estimation (Ng et al., 1999). The full E-step provides us with posteriors over actions, states and the total time. In practice we can use the α's in an efficient heuristic for pruning computations during inference (see section 1.3.3 and appendix .2). Further, the E-step generalizes to arbitrarily structured DBNs and thereby goes beyond standard Policy Evaluation, particularly when using approximate inference techniques like message passing or approximate belief representations.

Concerning the M-step, in the unstructured MDP the greedy M-step is *identical* to the policy update in Policy Iteration. That means that one iteration of the EM will yield exactly the same policy update as one iteration of Policy Iteration (provided one does exact inference and exact value function computation without time horizon cutoffs). Again, the M-step goes beyond a standard policy update in the generalized case. This becomes particularly apparent when in structured DBNs (e.g. the POMDP case in section 1.4) the full posteriors computed via inference (including forward propagated messages analogous to α's) are necessary for the M-step.

In summary,

Lemma 1. *The EM-algorithm on an unstructured MDP using exact inference and the greedy M-step is equivalent to Policy Iteration in terms of the policy updates performed.*

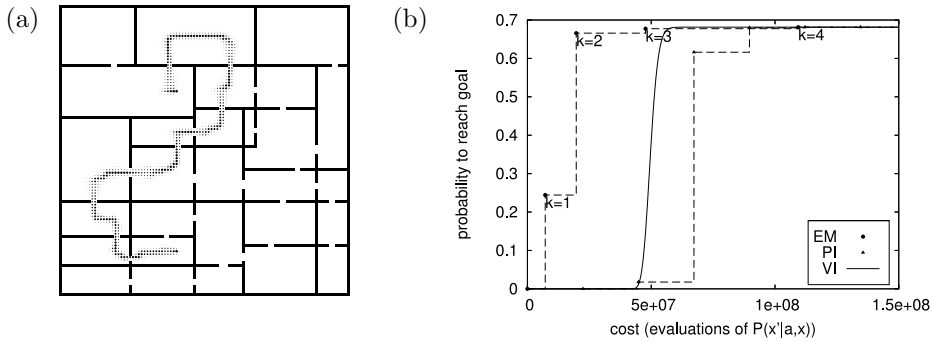

Figure 1.4: (a) State visiting probability calculated by EM for some start and goal state. The radii of the dots are proportional to $P(s \in \xi \,|\, R)$. (b) The probability of reaching the goal (for EM) and the value calculated for the start state (PS) against the cost of the planning algorithms (measured by evaluations of $P(s'|a, s)$).

Interestingly, this also means they are equivalent w.r.t. convergence. (Recall that Policy Iteration is guaranteed to converge to the global optimum whereas EM-algorithms are only guaranteed to converge to local optima.) The computational costs of both methods may differ depending on the implementation (see below).

Finally, the incremental E-step of Algorithm 2 only updates the β and α functions by propagating them for H steps. For $H = 1$ and when using the greedy M-step this is equivalent to Value Iteration. In a structured (but fully observable!) MDP, we have the same equivalence with structured Value Iteration.

1.3.3 Discrete maze examples

Efficiency. We first tested the EM algorithm with standard E-step and greedy M-step on a discrete maze of size 100×100 and compared it to standard Value Iteration (VI) and Policy Iteration (PI). Walls of the maze are considered to be trap states (leading to unsuccessful trials) and actions (north, south, east, west, stay) are highly noisy in that with a probability of 0.2 they lead to random transitions. In the experiment we chose a uniform time prior (discount factor $\gamma = 1$), initialized π uniformly, and iterated the policy update $k = 5$ times. To increase computational efficiency we exploited that the algorithm explicitly calculates posteriors which can be used to prune unnecessary computations during inference as explained in appendix .2. For policy evaluation in PI we performed 100 iterations of standard value function updates.

Figure 1.4(a) displays the posterior state visiting probabilities $P(s \in \xi \,|\, R)$ of the optimal policy computed by the EM for a problem where a reward of 1 is given when the goal state g is reached and the agent is initialized at a start state s. Computational costs are measured by the number of evaluations of the environment $P(s'|a, s)$ needed during the planning procedure. Figure 1.4(b) displays the probability of reaching the goal $P(R; \pi)$ against these costs. Note that for EM (and PI) we can give this information only after a complete E- and M-step cycle (policy evaluation and update) which are the discrete dots (triangles) in the graph. The graph also displays the curve for VI, where the currently calculated value V_A of the start state (which converges to $P(R)$ for the optimal policy) is plotted against how often VI evaluated $P(s'|a, s)$.

In contrast to VI and PI, the inference approach takes considerable advantage of knowing the start state in this planning scenario: the forward propagation allows for the pruning and the early decision on cutoff times in the E-step as described in

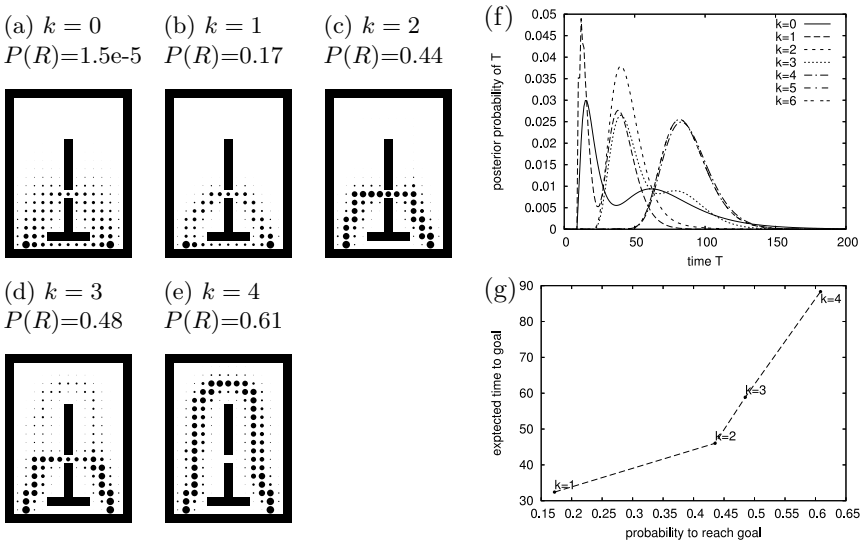

Figure 1.5: (a-e) State visiting probabilities for various EM-iterations k. The start and goal states are to the bottom left and right, respectively. The radii of the dots are proportional to $P(s \in \xi \,|\, R)$. (f) The different possible pathways lead to a multi-modal time posterior $P(T \,|\, R)$. (g) The trade-off between the expected time to goal (mean of $P(T \,|\, R)$) and the probability to reach the goal. The dots corresponds to $k = 1, .., 4$ (from left to right).

appendix .2. It should thus not surprise and not be overstated that the EM is more efficient in this specific scenario. Certainly, a similar kind of forward propagations could also be introduced for VI or PI to achieve equal efficiency. Nonetheless, our approach provides a principled way of pruning by exploiting the computation of proper posteriors. The policies computed by all three methods are equal for states which have significantly non-zero state visiting probabilities.

Multi-modal time posteriors. The total time T plays a special role as a random variable in our mixture model. We use another simple experiment to illustrate this special role by considering the total time posteriors. Modelling walls as trap states leads to interesting trade-offs between staying away from walls in favor of security and choosing short paths. Figure 1.5 displays a 15x20 maze with three possible pathways from the start (bottom left) to the goal (bottom right) state. The direct pathway is a narrow aisle clinched between two walls and thus highly risky. The next one up requires a step through a narrow doorway. The most top one is secure but longest. The five Figures illustrate the state visiting probability $P(s \in \xi \,|\, R)$ for random walks ($k = 0$) and the policies calculated by EM for $k = 1, .., 4$ iterations. Also the success probability $P(R)$ is indicated. Figure 1.5(f) displays the corresponding time posteriors $P(T \,|\, R)$ for the different k's. Interesting is the multi-modality of these time posteriors in that specific environment. The multi-modality in some way reflects the topological properties of the environment: that there exists multiple possible pathways from the start to the goal with different typical lengths (maxima of the time posterior) and different success probabilities (area (integral) of a mode of the time posterior). Already for $k = 0$ the multi-modality exhibits that, besides the direct pathway (of typical length ≈ 15), there exist alternative, longer routes which comprise significant success probability. One way to exploit this insight could be to choose a new time prior for the next inference iteration that explicitly favors these longer routes. Figure 1.5(g) nicely exhibits the

trade-off between the expected time to goal and the probability to reach the goal.

1.3.4 Stochastic optimal control

Gaussian belief state propagation. Next we want to show that the framework naturally allows to transfer other inference techniques to the problem of solving MDPs. We address the problem of stochastic optimal control in the case of a continuous state and control space. A standard inference technique in continuous state spaces is to assume Gaussian belief states as representations for $\boldsymbol{a}$'s and $\boldsymbol{b}$'s and propagate forward-backward and using the unscented transform to handle also non-linear transition dynamics (see (Murphy, 2002) for an overview on inference techniques in DBNs). Note that using Gaussian belief states implies that the effective value function (section 1.3.2) becomes a mixture of Gaussians.

All the equations we derived remain valid when reinterpreted for the continuous case (summations become integrations, etc) and the exact propagation equations (1.37) and (1.40) are replaced by propagations of Gaussian belief states using the unscented transform. In more detail, let $\mathcal{N}(x, a, A)$ be the normal distribution over x with mean a and covariance A and let $\overline{\mathcal{N}}(x, a, A)$ be the respective *non-normalized* Gaussian function with $\overline{\mathcal{N}}(a, a, A) = 1$. As a transition model we assume

$$P(x'|u, x) = \mathcal{N}(x', \phi(u, x), Q(u)) \ , \quad Q(u) = C + \mu|u|^2 \ I \tag{1.52}$$

where $\phi(u, x)$ is an non-linear function depending on the current state x and the control signal u, C is a constant noise covariance, and we introduced a parameter μ for an additional noise term that is squared in the control signal. With the parameterization $\boldsymbol{a}_t(x) = \mathcal{N}(x, a_t, A_t)$ and $\boldsymbol{b}_\tau(x) = \overline{\mathcal{N}}(x, b_\tau, B_\tau)$ (note that $\boldsymbol{b}$'s always remain non-normalized Gaussian likelihoods during propagation), forward and backward propagation read

$$(a_t, A_t) = UT_\phi(a_{t-1}, A_{T-1}) \tag{1.53}$$

$$(b_\tau, B_\tau) = UT_{\phi^{-1}}(b_{\tau-1}, B_{\tau-1}) \ , \tag{1.54}$$

where $UT_\phi(a, A)$ denotes the unscented transform of a mean and covariance under a non-linear function. In brief, this transform deterministically considers $2n+1$ points (say with standard deviation distance to the mean) representing the Gaussian. In the forward case (the backward case) it maps each point forward using ϕ (backward using ϕ^{-1}), associates a covariance $Q(u)$ (a covariance $\phi'^{-1} \ Q(u) \ \phi'^{-1T}$, where ϕ'^{-1} is the local inverse linearization of ϕ at each point) with each point, and returns the Gaussian that approximates this mixture of Gaussians. Further, for any t and τ we have

$$P(R \,|\, T{=}t+\tau) = \mathcal{N}(a_t, b_\tau, A_t + B_\tau) \tag{1.55}$$

$$P(x_t{=}x \,|\, R, T{=}t+\tau) = \mathcal{N}(x, c_{t\tau}, C_{t\tau}) \ ,$$

$$C_{t\tau}^{-1} = A_t^{-1} + B_\tau^{-1} \ , \quad c_{t\tau} = C_{t\tau} \ (A_t^{-1} \ a_t + B_\tau^{-1} \ b_\tau) \tag{1.56}$$

$$\tag{1.57}$$

The policy and the M-step. In general, the policy is given as an arbitrary non-linear function $\pi : \ x \mapsto u$. Clearly, we cannot store such a function in memory. However, via the M-step the policy can always be implicitly expressed in terms of the $\boldsymbol{b}$-quantities of the previous E-step and numerically evaluated at specific states x. This is particularly feasible in our case because the unscented transform used in the belief propagation (of the next E-step) only needs to evaluate the transition

function ϕ (and thereby π) at some states; and we have the advantage of not needing to approximate the function π in any way. For the M-step (1.51) we need to maximize the mixture of Gaussians (see (1.35))

$$\hat{q}_\tau(u,x) := \Big[P(R|u,x) + \int_{x'} P(x'|u,x)\ \beta(x')\Big] \tag{1.58}$$

$$\beta(x') = \frac{1}{1-\gamma}\sum_{\tau=0}^{\infty} P(T{=}\tau+1)\ \overline{\mathcal{N}}(x', b_{\tau-1}, B_{\tau-1}) \tag{1.59}$$

We use a gradient ascent. The gradient for each component of the mixture of Gaussians is:

$$q_\tau(u,x) := \int_{x'} P(x'|u,x)\ \overline{\mathcal{N}}(x', b_{\tau-1}, B_{\tau-1}) \tag{1.60}$$

$$= |2\pi B_{\tau-1}|^{1/2}\ \mathcal{N}(b_{\tau-1}, \phi(u,x), B_{\tau-1} + Q(u)) \tag{1.61}$$

$$\partial_u q_\tau(u,x) = -q_\tau(u,x)\Big[\ h^T\Big(\partial_u\phi(u,x)\Big) - \mu u\Big(\mathrm{tr}(A^{-1}) - h^T h\Big)\Big]$$

$$A := B_{\tau-1} + Q(u)\ , \quad h := A^{-1}\ (\phi(u,x) - b) \tag{1.62}$$

We perform this gradient ascent whenevery we query the policy at a specific state x.

Examples. Consider a simple 2-dimensional problem where the start state is distributed around zero via $\boldsymbol{a}_0(x) = \mathcal{N}(x,\ (0,0),\ .01I)$ and the goal region is determined by $P(R\,|\,x) = \overline{\mathcal{N}}(x,\ (1,1),\ \mathrm{diag}(.0001, .1))$. Note that this goal region around $(1,1)$ is heavily skewed in that rewards depend more on the precision along the $x-$dimension than the $y-$dimension. We first consider a simple control law $\phi(u,x) = x + .1\,u$ and the discount factor $\gamma = 1$. When choosing $\mu = 0$ (no control-dependent noise), the optimal control policy will try to jump directly to the goal $(1,1)$. Hence we first consider the solution when manually constraining the norm of $|u|$ to be small (effectively following the gradient of $P(r=1\,|\,u_t=u, x_t=x;\pi)$). Figure 1.6(a,b) shows the learned control policy π and the forward simulation given this policy by displaying the covariance ellipses for $\boldsymbol{a}_{0:T}(x)$ after $k=3$ iterations. What we find is a control policy that reduces errors in $x-$dimension more strongly than in $y-$direction, leading to the tangential approach to the goal region. This is related to studies on redundant control or the so-called uncontrolled manifold.

Next we can investigate what the effect of control-dependent noise is without a constraint on the amplitude of u. Figure 1.6(c,d) displays results (after $k=3$ iterations) for $\mu = 1$ and no additional constraints on u. The process actually resembles a golf player: the stronger the hit, the more noise. The optimal strategy is to hit fairly hard in the beginning, hopefully coming closer to the goal, such that later a number of smaller and more precise hits can be made. The reason for the small control signals around the goal region is that small steps have much more accuracy and reward expectation is already fairly large for just the $x-$coordinate being close to 1.

Finally we think of x being a phase space and consider the dynamics $\phi(u,x) = (x_1 + .1x_2, x_2 + .1u)$ where u is the 1-dimensional acceleration of the velocity x_2, and x_1 is a position. This time we set the start and goal to $(0,0)$ and $(1,0)$ respectively, both with variance .001 and choose $\mu = 10$. Figure 1.6(e,f) display the result and show nicely how the learned control policy approaches the new position on the $x-$axis by first gaining and then reducing velocity.

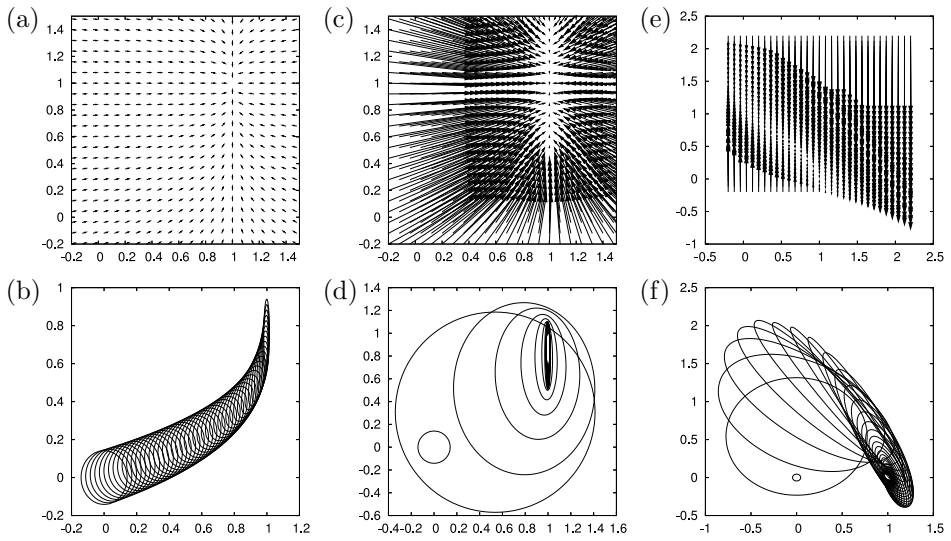

Figure 1.6: Learned policies (left) and forward simulation ($\boldsymbol{a}$'s) of these policies (right) for aspheric Gaussian-shaped targets. (a,b) are for the case of restricted action amplitude (the walker model). (c,d) are for unconstrained amplitude (the golfer model). And (e,f) are for the approach to a new position under phase space dynamics.

1.4 Application to POMDPs

A stationary, partially observable Markov decision process (POMDP, see e.g. (Kaelbling et al., 1998)) is given by four time-independent probability functions,

the initial world state distribution	$P(s_0 = s)$
the world state transitions	$P(s_{t+1} = s' \mid a_t = a, s_t = s)$
the observation probabilities	$P(y_t = y \mid s_t = s)$
the reward probabilities	$P(r_t = r \mid a_t = a, s_t = s)$.

These functions are considered known. We assume the world states, actions, and observations (s_t, y_t, a_t) are discrete random variables while the reward r_t is a real number.

The POMDP only describes one "half" of the process to be described as a DBN — the other half is the agent interacting with the environment. Our point of view is that the agent could use an arbitrary "internal machinery" to decide on actions. FSCs are a simple example. However, a general DBN formulation of the agent's internal machinery allows us to consider much more structured ways of behavior organization, including factorized and hierarchical internal representations (see, e.g., Theocharous et al., 2004; Toussaint et al., 2008). In the remainder of this section we investigate a policy model that is slightly different to finite state controllers but still rather simple. However, the approach is generally applicable to any DBN formulation of the POMDP and the agent.

To solve a given POMDP challenge an agent needs to maintain some internal memory variable (if not the full belief state) that represents information gained from previous observations and actions. We assume that this variable is updated depending on the current observation and used to *gate* reactive policies rather than to directly emit actions. More precisely, the dynamic Bayesian network in Figure

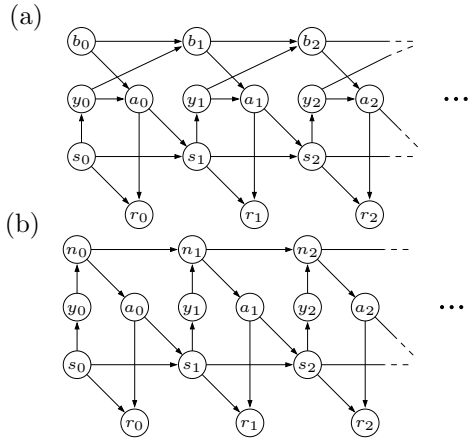

Figure 1.7: (a) DBN of the POMDP and policy with internal memory b_t; the time is unbounded and rewards are emitted at every time step. (b) For comparison: the DBN of a POMDP with a standard FSC, with "node state" n_t.

1.7(a) captures the POMDP and the agent model which is defined by the

initial internal memory distribution	$P(b_0 = b) =: \nu_b$
internal memory transition	$P(b_{t+1} = b' \mid b_t = b, y_t = y) =: \lambda_{b'by}$
reactive policies	$P(a_t = a \mid b_t = b, y_t = y) =: \pi_{aby}$.

Here we introduced b_t as the agent's internal memory variable. It is comparable to the "node state" in finite state controllers (Figure 1.7(b)), but differs in that it does not directly emit actions but rather gates reactive policies: for each internal memory state b the agent uses a different "mapping" π_{aby} (i.e. a different reactive policy) from observations to actions.

As for the MDP case, solving the POMDP in this approach means to find parameters $\theta = (\nu, \lambda, \pi)$ of the DBN in Figure 1.7 that maximize the expected future return $V^\theta = \mathrm{E}\{\sum_{t=0}^{\infty} \gamma^t \, r_t; \theta\}$ for a discount factor $\gamma \in [0, 1)$.

M-step. The M-steps for the parameters can directly be derived from the free energy in the form (1.47). We have:

$$\pi^{\text{new}}_{aby} = \frac{\pi^{\text{old}}_{aby}}{C_{by}} \sum_s \Big[P(R|a,s) \; + \; \sum_{b's'} \beta(b',s') \; \lambda_{b'by} \; P(s'|a,s) \Big] \; P(y|s) \; \alpha(b,s) \; , \tag{1.63}$$

$$\lambda^{\text{new}}_{b'by} = \frac{\lambda^{\text{old}}_{b'by}}{C'_{by}} \sum_{s',a,s} \beta(b',s') \; P(s'|a,s) \; \pi_{aby} \; P(y|s) \; \alpha(b,s) \; , \tag{1.64}$$

$$\nu^{\text{new}}_b = \frac{\nu^{\text{old}}_b}{C''_b} \sum_x \beta(b,s) \; P(s_0 = s) \; , \tag{1.65}$$

where C_{by}, C'_{by} and C''_b are normalization constants. Note that in these updates the α's (related to state visiting probabilities) play a crucial role. Also we are not aware of a greedy version of these updates that proved efficient (i.e. without immediate convergence to a local minimum).

Complexity. Let S, B, A and Y momentarily denote the cardinalities of random variables s, b, a, y, respectively. The main computational cost accumulates during $\boldsymbol{a}$- and $\boldsymbol{b}$-propagation; with the separator (b_t, s_t) both of which have complexity $O(H\, B^2\, S^2)$. Here and below, S^2 scales with the number of non-zero elements in the transition matrix $P(s'|s)$ (assuming non-zero action probabilities). We always use sparse matrix representations for transition matrices. The number of propagations H scales with the expected time of reward (for a simple start-goal scenario this is the expected time to goal). Sparse vector representations of α's and β's further reduce the complexity depending on the topological dimensionality of $P(s'|s)$. The computational complexity of the M-step scales with $O(AY B^2 S^2)$; in total this adds to $O((H + AY)B^2S^2)$ for one EM-iteration.

For comparison, let N denote the number of nodes in a FSC. The computation of a policy gradient w.r.t. a *single* parameter of a FSC scales with $O((H + AY)N^2S^2)$ (taken from (Meuleau et al., 1999), top of page 7). A fully parameterized FSC has $NA + N^2Y$ parameters, bearing a total complexity of $O((H + AY)N^4S^2Y)$ to compute a full policy gradient.

For EM-learning as well as gradient ascent, the complexity additionally multiplies with the number k of EM-iterations respectively gradient updates.

1.4.1 POMDP experiments

Scaling. The POMDP EM-algorithm has no free parameters except for the initializations of $\lambda_{b'by}$, π_{aby}, and ν_b. Roughly, we initialized ν_b and π_{aby} approximately uniformly, while $\lambda_{b'by}$ was initialized in a way that favors not to switch the internal memory state, i.e., the diagonal of the matrix $\lambda_{b'b\cdot}$ was initialized larger than the off-diagonal terms. More precisely, we first draw non-normalized numbers

$$\pi_{aby} \sim 1 + 0.1\,\mathcal{U}([0,1]) \;, \quad \lambda_{b'by} \sim 1 + 5\,\delta_{b'b} + 0.1\,\mathcal{U}([0,1]) \;, \quad \nu_b = 1 \qquad (1.66)$$

where $\mathcal{U}([0,1])$ is the uniform distribution over $[0,1]$, and then normalize these parameters.

To start with, we test the scaling behavior of our EM-algorithm and compare it with that of gradient ascent for a FSC (FSC-GA). We tried three options for coping with the problem that the simple policy gradient in (Meuleau et al., 1999) ignores the normalization constraints of the parameters: (1) projecting the gradient on the simplex, (2) using a step-size-adaptive gradient ascent (RPROP) with added soft-constraint gradients towards the simplex, (3) using MATLAB's gradient-based constraint optimization method 'fmincon'. The second option gave the best results and we refer to those in the following. Note that our algorithm does not have such problems: the M-step assigns correctly normalized parameters. Figure 1.8 displays the results for the simple maze considered in (Meuleau et al., 1999) for various maze sizes. Our policy model needs $B = 2$ internal memory states, the FSC $N = 5$ graph nodes to solve these problems. The discount factor was chosen $\gamma = .99$. The results confirm the differences we noticed in the complexity analysis.

Training the memory to gate primitive reactive behaviors. To exemplify the approach's ability to learn an appropriate memory representation for a given task we investigate further maze problems. We consider a *turtle*, which can move forward, turn right or left, or wait. With probability $1-\epsilon$ this action is successful; with probability $\epsilon = .1$ the turtle does not respond. The state space is the cross product of positions and four possible orientations, and the observations are a 4 bit number encoding the presence of adjacent walls relative to the turtle's orientation.

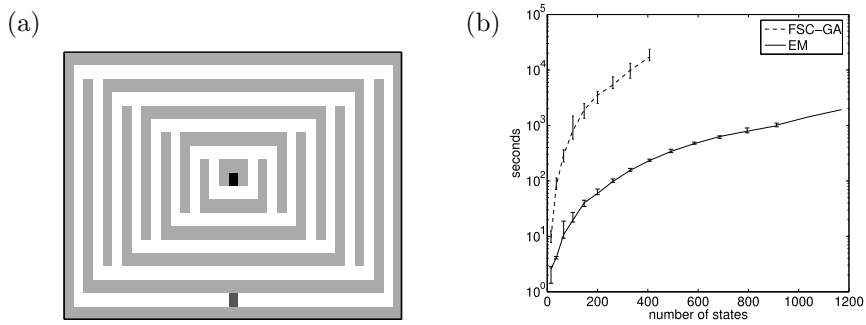

Figure 1.8: (a) A simple scalable maze from (Meuleau et al. 1999), here with 330 states. The start (goal) position is marked gray (black). The robot has five actions (north, south, eat, west, stay) and his observation is a 4 bit number encoding the presence of adjacent walls. (b) Running times of EM-learning and FSC gradient ascent that show how both methods scale with the maze size. The lines are medians, the errorbars min and max of 10 independent runs for each of the various maze sizes. (For maze sizes beyond 1000 we display only 1 run).

Further, whenever the agent reaches the goal (or a zero-reward drain state, see below) it is instantly reset to the start position.

Figures 1.9(a) and 1.9(b) display two small mazes with two specific difficulties: The interior states and also the entries and exits (cross-shaded) of halls in 1.9(a) all have the same observation 0000 and are highly susceptible for the agent to get lost. For $B = 2$ (that is, when the latent variable b has two possible values), the turtle learns a wall-following strategy as a basic reactive behavior, while the internal memory is used only at the exists and entrances to halls: for internal state $b = 1$ and observation $y =$ 0000 the turtle turns left and switches to $b = 2$, while for $b = 2$ and $y =$ 0000 the turtle goes straight and switches back to $b = 1$. The maze in Figure 1.9(b) is a binary-decision maze and poses the problem of remembering how many junctions have passed already: To reach the goal, the turtle has to follow aisles and at T-junctions make decisions [left, right, right, left]. For $B = 3$ the algorithm finds the obvious solution: Each internal memory state is associated with simple reactive behaviors that follows aisles and, depending on b, turn left or right at a T-junction. A finite state controller would certainly find a very similar solution. However, in our case this solution generalizes to situations when the corridors are not straight: Figure 1.9(c, top left) displays a maze with 30 locations (number of states is 480), where the start state is in the top left corner and the goal state in the bottom right. Again, the turtle has to make decisions [left, right, right, left] at T-junctions to reach the goal, but additionally has to follow complex aisles in between. Unlike with FSCs, our turtle needs again only $B = 3$ internal memory states to represent the current corridor. The shading in Figure 1.9(c) displays the probability of visiting a location on a trajectory while being in memory state $b = 1, 2$ or 3.

Finally, we investigate the maze in Figure 1.9(d) with 379 locations (1516 turtle states). The maze is a complex combination of corridors, rooms and junctions. On this maze we also tested a normal robot (north, south, west, east actions with noise $\epsilon = .1$), learning curves for $B = 3$ for the left and right most goals are given in Figure 1.10(a,b) and exhibit reliable convergence.[4] We also investigated single runs

[4] As a performance measure we define the *expected reward interval* which is directly linked to $P(R)$. Consider a cyclic process that receives a reward of 1 every d time steps; the expected future reward of this process is $P(R) = \sum_{T=1}^{\infty} P(dT) = (1-\gamma)\sum_{T=1}^{\infty} \gamma^{dT} = \frac{\gamma^d\ (1-\gamma)}{1-\gamma^d}$. Inverting

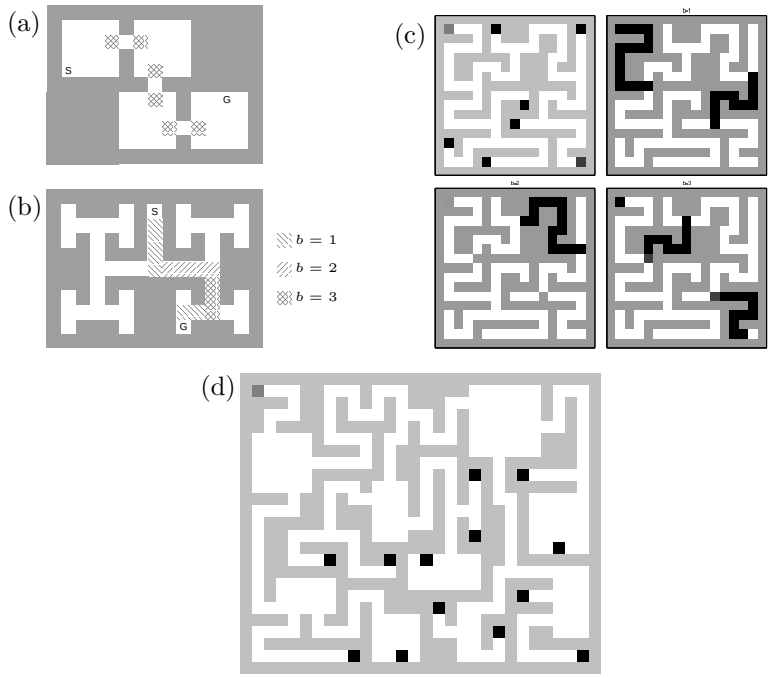

Figure 1.9: Further mazes considered in the experiments. (c): Top left is the maze with the start (light gray), goal (bottom right) and drain-states (dark). The other three illustrations display the internal memory state b (grey value $\propto \widehat{\alpha}(b,s)$ for $b = 1,2,3$) at different locations in the maze. (d): Large maze with 1516 turtle states.

in the turtle case for the left most goal (Figure 1.10(c)) and the second left goal (Figure 1.10(d) dashed line). Again, the turtle utilizes that aisle following can be implemented with a simple reactive behavior; the internal memory is only used for decision making in halls and at junctions. Since the aisle following behavior can well be generalized to all goal settings we performed another experiment: we took the final policy π_{aby} learned for the left most goal as an initialization for the task of finding the second left goal. Note that the start-to-goal paths are largely disjoint. Still, the algorithm converges, particularly in the beginning, much faster (Figure 1.10(d) solid line), showing that such generalization is indeed possible.

To conclude these experiments, we can summarize that the agent learned internal memory representations to switch between reactive behaviors. In the experiments they mainly turned out to represent different corridors. Generalization of the reactive behaviors to new goal situations is possible. Further, memorization is not time bounded, e.g., independent of the length of an aisle the turtle agent can sustain the current internal memory while executing the reactive aisle following behavior.

1.5 Conclusion

We introduced a framework for solving (PO)MDPs by translating the problem of maximizing expected future return into a problem of likelihood maximization. One ingredient for this approach is the mixture of finite-time models we introduced

this relation, we translate a given expected future reward into an expected reward interval via $d = \frac{\log P(R) - \log(P(R)+1-\gamma)}{\log \gamma}$. This measure is rather intuitive: The performance can directly be compared with the shortest path length to the goal. Note though that in stochastic environment even an optimal policy has an expected reward interval larger than the shortest path to goal.

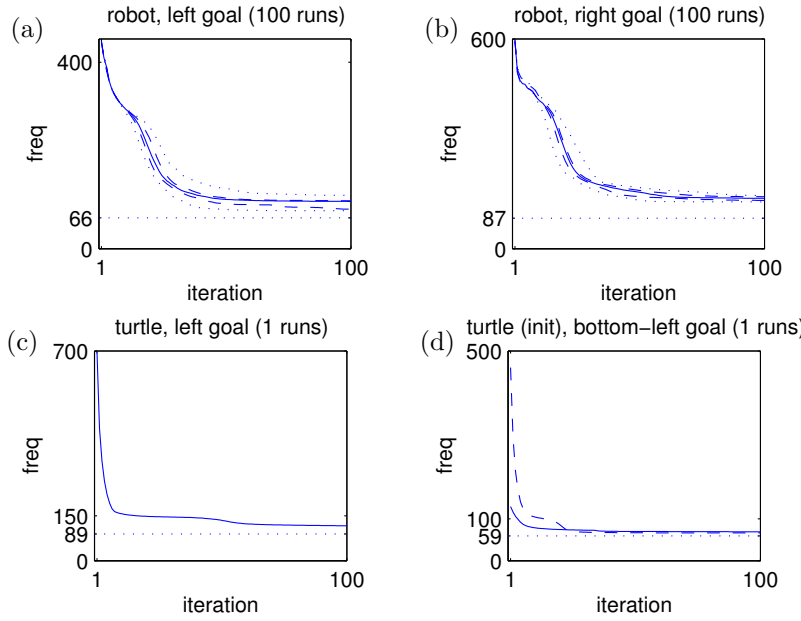

Figure 1.10: Learning curves for the maze in Figure 1.9(d). The expected reward interval (see footnote) is given over the number of EM-iterations. Solid: median over 100 independent trial runs (with noisy initializations (1.66)), dashed and dotted: 2.5, 25, 75 and 97.5 percentiles, dotted baseline: shortest path to the goal in the respective environment which *were* possible in the noise-free case. The optimal controller in our stochastic case is necessarily above this baseline.

in section 1.1. We have shown that this approach establishes equivalence for arbitrary reward functions, allows for an efficient inference procedure, propagating synchronously forward and backward without pre-fixing a finite time horizon H, and allows for the handling of discounting rewards. We also showed that in the case of an unstructured MDP the resulting EM-algorithm using exact inference and a greedy M-step is closely related to standard Policy Iteration.

However, unlike Policy Iteration, the EM-algorithm generalizes to arbitrary DBNs and the aim of this approach is to transfer the full variety of existing inference techniques to the problem of solving (PO)MDPs. This refers especially to structured problem domains, where DBNs allow us to consider structured representations of the environment (the world state, e.g. factorization or hierarchies) as well as the agent (e.g. hierarchical policies or multiple agents). Inference techniques like variational approaches, message-passing algorithms, or approximate belief representations in DBNs can be used to exploit such structure in (PO)MDPs.

1.5.1 Follow-up and related work

We exemplified the approach for exact inference on unstructured MDPs, using Gaussian belief state propagation on a non-linear stochastic optimal control problem, and on a more complex DBN formulation of a POMDP problem. Recently there have be a series of papers based on or closely related to the general framework that we presented here. In (Toussaint et al., 2008) we extended the approach to learning hierarchical controllers for POMDPs. In (Vlassis and Toussaint, 2009) we presented a model-free Reinforcement Learning version of our EM approach. Hoffman et al. (2008, 2009b) uses MCMC methods for approximate inference in this context and

generalize the EM algorithm for continuous MDPs (Hoffman et al., 2009a). Finally, Peters and Schaal (2007); Kober and Peters (2009) developed similar EM techniques in a robotics and model-free context.

An interesting issue for future research is to consider max-product BP (a generalization of Viterbi) in the context of planning. In the POMDP context, further aspects to consider are: Can we use inference techniques also to estimate the number of internal states we need to solve a problem (cf. Infinite Hidden Markov models (Beal et al., 2002) as a method to learn the number of hidden states needed to model the data)? Or are there efficient heuristics to add hidden states in a DBN, e.g., analogous to how new nodes are added to FSCs in the Bounded FSCs approach (Poupart and Boutilier, 2004)?

We hope that our approach lays new ground for a whole family of new, inference-based techniques being applicable in realm of (PO)MDPs.

Acknowledgments

M.T. is grateful to the German Research Foundation (DFG) for the Emmy Noether fellowship TO 409/1-3.

.1 Remarks

(i) The mixture of finite-time MDPs may be compared to a classical interpretation of reward discounting: Assume the agent has a probability $(1-\gamma)$ of dying after each time step. Then the distribution over his life span is the geometric distribution $P(T) = \gamma^T(1-\gamma)$. In our mixture of finite-time MDPs we treat each possible life-span T separately. From the agent's perspective, he knows that he has a finite life span T but he does not know what it is – he lives in a mixture of possible worlds. Each finite life span is terminated by a single binary reward (say, going to heaven or hell). The agent's behavior must reflect his uncertainty about his life span and act by accounting for the probability that he might die now or later on, i.e., he must "average" over the mixture of possible worlds he might live in.

(ii) In the original MDP, the rewards at two different time slices, say r_t and r_{t+1}, are strongly correlated. The mixture of finite-time MDPs does not include such correlations because the observations of reward at $T=t$ and $T=t+1$ are treated by separate finite-time MDPs. However, since the expected future return V^π is merely a *sum* of reward expectations at different time slices such correlations are irrelevant for solving the MDP and computing optimal policies.

(iii) Do exponentiated rewards as observation likelihoods lead to equivalence? Let us introduce modified binary reward variables $\hat{r}_t$ in every time slice with probabilties

$$P(\hat{r}_t=1 \,|\, a_t, s_t) = e^{\gamma^t \mathcal{R}(a_t,s_t)} \;, \quad \mathcal{R}(a_t,s_t) := \mathrm{E}\{r_t \,|\, a_t, s_t\} \;. \tag{67}$$

Then

$$\log P(\hat{r}_{0:T}=1;\pi) = \log \mathrm{E}_{a_{0:T},s_{0:T}}\{\prod_{t=0}^{T} P(\hat{r}_t=1 \,|\, a_t, s_t)\} \tag{68}$$

$$\geq \mathrm{E}_{a_{0:T},s_{0:T}}\{\log \prod_{t=0}^{T} P(\hat{r}_t=1 \,|\, a_t, s_t)\} \tag{69}$$

$$= \mathrm{E}_{a_{0:T},s_{0:T}}\{\sum_{t=0}^{T} \gamma^t \mathcal{R}(a_t,s_t)\} \tag{70}$$

$$= V^\pi \tag{71}$$

That is, maximization of $P(\hat{r}_{0:T}=1;\pi)$ is *not* equivalent to maximization of V^π. However, this points to a formulation in terms of a Kullback-Leibler divergence minimization: We have

$$V^\pi = \mathrm{E}_{a_{0:T},s_{0:T}}\{\log \prod_{t=0}^{T} P(\hat{r}_t=1 \,|\, a_t, s_t)\} \tag{72}$$

$$= \sum_{a_{0:T},s_{0:T}} \Big[P(s_0)\ \pi(a_0 \,|\, s_0)\ \prod_{t=1}^{T} \pi(a_t \,|\, s_t)\ P(s_t \,|\, a_{t\text{-}1}, s_{t\text{-}1})\Big]$$

$$\cdot \log \Big[\prod_{t=0}^{T} \exp\{\gamma^t \mathcal{R}(a_t, s_t)\Big] \tag{73}$$

$$= \sum_{a_{0:T},s_{0:T}} p(a_{0:T}, s_{t:T} \,|\, \pi) \log q(a_{0:T}, s_{t:T}) \tag{74}$$

where we defined

$$p(a_{0:T}, s_{t:T} \,|\, \pi) = P(s_0)\ \pi(a_0 \,|\, s_0)\ \prod_{t=1}^{T} \pi(a_t \,|\, s_t)\ P(s_t \,|\, a_{t\text{-}1}, s_{t\text{-}1}) \tag{75}$$

$$q(a_{0:T}, s_{t:T}) = \prod_{t=0}^{T} \exp\{\gamma^t \mathcal{R}(a_t, s_t)\} \tag{76}$$

The first distribution p is the prior trajectory defined by the policy π disregarding any rewards. The second "distribution" (if one normalizes it) $q(a_{0:T}, s_{0:T})$ has a very simple form, it fully factorizes over time and in each time slice we have the exponentiated reward with "temperature" γ^{-t}. If $q(s_{0:T})$ is normalized, we can also write the value in terms of a Kullback-Leibler divergence,

$$V^\pi = -D\big(p \,\big\|\, q\big) + H(p)\ . \tag{77}$$

.2 Pruning computations

Consider a finite state space and assume that we fixed the maximum allowed time T by some upper limit T_M (e.g., by deciding on a cutoff time based on the time posterior computed on the fly, see below). Then there are potentially large regions of the state space on which we may prune computations, i.e., states s for which the posterior $P(s_t = s \,|\, T = t+\tau) = 0$ for any t and τ with $t+\tau \le T_M$. Figure 11 illustrates the idea. Let us consider the $\boldsymbol{a}$-propagation (1.37) first (all statements apply conversely for the $\boldsymbol{b}$-propagation). For iteration time t we define a set of states

$$S_a(t) = \{s \,|\, a_t(s) \neq 0\ \wedge\ (t < T_M/2\ \vee\ b_{T_M-t}(s) \neq 0)\}\ . \tag{78}$$

Under the assumption that $b_\tau(s)=0 \Rightarrow \forall \tau' \le \tau : b_{\tau'}(s) = 0$ it follows

$$\begin{aligned} i \in S_a(t) &\Leftarrow\ a_t(s) \neq 0\ \wedge\ b_{T_M-t}(s) \neq 0 \\ &\Leftarrow\ \exists_{\tau \le T_M-t} :\ a_t(s) \neq 0\ \wedge\ b_\tau(s) \neq 0 \\ &\iff\ \exists_{\tau \le T_M-t} :\ \gamma_{t\tau}(s) \neq 0 \end{aligned} \tag{79}$$

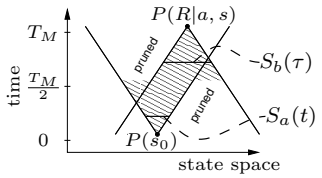

Figure 11: Only the envelopes emanating from the start distribution ($P(s)$) and rewards ($P(R|a,s)$) contribute to the propagation. They are chopped where they do not overlap with the other envelope after $T_M/2$ iterations.

Thus, every state that is potentially visited at time t (for which $\exists_{\tau:t+\tau\leq T_M} : \gamma_{t\tau}(s) \neq 0$) is included in $S_a(t)$. We will exclude all states $s \notin S_a(t)$ from the a-propagation procedure and not deliver their messages. The constraint $t < T_M/2$ concerning the b's was inserted in the definition of $S_a(t)$ only because of the feasibility of computing $S_a(t)$ at iteration time t. Initializing $S_a(0) = \{s \,|\, P(s) \neq 0\}$, we can compute $S_a(t)$ recursively via

$$S_a(t) = \begin{cases} S_a(t-1) \cup \mathsf{OUT}(S_a(t-1)) & \text{for } t < T_M/2 \\ \Big[S_a(t-1) \cup \mathsf{OUT}(S_a(t-1))\Big] \cap \{s \,|\, b_{T_M-t}(s) \neq 0\} & \text{for } t \geq T_M/2 \end{cases}, \tag{80}$$

where $\mathsf{OUT}(S_a(t-1))$ is the set of states which have non-zero probability transitions from states in $S_a(t-1)$. Analogously, the book keeping for states that participate in the b-propagation is

$$S_b(0) = \{s \,|\, P(R|a,s) \neq 0\} \tag{81}$$

$$S_b(\tau) = \begin{cases} S_b(\tau-1) \cup \mathsf{IN}(S_b(\tau-1)) & \text{for } \tau < T_M/2 \\ \Big[S_b(\tau-1) \cup \mathsf{IN}(S_b(\tau-1))\Big] \cap \{s \,|\, a_{T_M-\tau}(s) \neq 0\} & \text{for } \tau \geq T_M/2 \end{cases}. \tag{82}$$

For the discount prior, we can use a time cutoff T_M for which we expect further contributions to be insignificant. The choice of this cutoff involves a payoff between computational cost and accuracy of the E-step. Let T_0 be the minimum T for which the finite-time likelihood $P(R \,|\, T; \pi) \neq 0$. It is clear that the cutoff needs to be greater than T_0. In the experiment in section 1.3.3 we used an increasing schedule for the cutoff time, $T_M = (1 + 0.2\,k)\,T_0$, depending on the iteration k of the EM-algorithm to ensure that with each iteration we become more accurate.

Bibliography

Atkeson, C. and Santamaría, J. (1997). A comparison of direct and model-based reinforcement learning. In *Int. Conf. on Robotics and Automation*.

Attias, H. (2003). Planning by probabilistic inference. In Bishop, C. M. and Frey, B. J., editors, *Proc. of the 9th Int. Workshop on Artificial Intelligence and Statistics*.

Beal, M. J., Ghahramani, Z., and Rasmussen, C. E. (2002). The infinite hidden markov model. In Dietterich, T., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*. MIT Press.

Boutilier, C., Dean, T., and Hanks, S. (1999). Decision theoretic planning: structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94.

Boutilier, C., Dearden, R., and Goldszmidt, M. (1995). Exploiting structure in policy construction. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI 1995)*, pages 1104–1111.

Bui, H., Venkatesh, S., and West, G. (2002). Policy recognition in the abstract hidden markov models. *Journal of Artificial Intelligence Research*, 17:451–499.

Chavira, M., Darwiche, A., and Jaeger, M. (2006). Compiling relational bayesian networks for exact inference. *Int. Journal of Approximate Reasoning*, 42:4–20.

Cooper, G. (1988). A method for using belief networks as influence diagrams. In *Proc. of the Fourth Workshop on Uncertainty in Artificial Intelligence*, pages 55–63.

Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research (JAIR)*, 19:399–468.

Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., and Boutilier, C. (1998). Hierarchical solution of Markov decision processes using macro-actions. In *Proc. of Uncertainty in Artificial Intelligence (UAI 1998)*, pages 220–229.

Hoffman, M., de Freitas, N., Doucet, A., and Peters, J. (2009a). An expectation maximization algorithm for continuous markov decision processes with arbitrary rewards. In *Twelfth Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2009)*.

Hoffman, M., Doucet, A., de Freitas, N., and Jasra, A. (2008). Bayesian policy learning with trans-dimensional MCMC. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*. MIT Press.

Hoffman, M., Kueck, H., Doucet, A., and de Freitas, N. (2009b). New inference strategies for solving markov decision processes using reversible jump mcmc. In *Uncertainty in Artificial Intelligence (UAI 2009)*.

Kaelbling, L., Littman, M., and Moore, A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134.

Kober, J. and Peters, J. (2009). Policy search for motor primitives in robotics. In Koller, D., Schuurmans, D., and Bengio, Y., editors, *Advances in Neural Information Processing Systems 21*. MIT Press, Cambridge, MA.

Koller, D. and Parr, R. (1999). Computing factored value functions for policies in structured MDPs. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI 1999)*, pages 1332–1339.

Kveton, B. and Hauskrecht, M. (2005). An MCMC approach to solving hybrid factored MDPs. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, volume 19, pages 1346–1351.

Littman, M. L., Majercik, S. M., and Pitassi, T. (2001). Stochastic boolean satisfiability. *Journal of Automated Reasoning*, 27(3):251–296.

McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proc. of the 18th Int. Conf. on Machine Learning (ICML 2001)*, pages 361–368.

Meuleau, N., Peshkin, L., Kim, K.-E., and Kaelbling, L. P. (1999). Learning finite-state controllers for partially observable environments. In *Proc. of Fifteenth Conf. on Uncertainity in Artificial Intelligence (UAI 1999)*, pages 427–436.

Minka, T. (2001). A family of algorithms for approximate bayesian inference. PhD thesis, MIT.

Murphy, K. (2002). Dynamic bayesian networks: Representation, inference and learning. PhD Thesis, UC Berkeley, Computer Science Division.

Ng, A. Y., Parr, R., and Koller, D. (1999). Policy search via density estimation. In *Advances in Neural Information Processing Systems*, pages 1022–1028.

Pearl, J. (1988). *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Peters, J. and Schaal, S. (2007). Reinforcement learning by reward-weighted regression for operational space control. In *Proc. of the Int. Conf. on Machine Learning (ICML 2007)*.

Poupart, P. and Boutilier, C. (2004). Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16 (NIPS 2003)*, volume 16. MIT Press.

Raiko, T. and Tornio, M. (2005). Learning nonlinear state-space models for control. In *Proc. of Int. Joint Conf. on Neural Networks (IJCNN 2005)*.

Shachter, R. D. (1988). Probabilistic inference and influence diagrams. *Operations Research*, 36:589–605.

Theocharous, G., Murphy, K., and Kaelbling, L. (2004). Representing hierarchical POMDPs as DBNs for multi-scale robot localization. In *Intl. Conf. on Robotics and Automation (ICRA 2004)*.

Toussaint, M. (2009). Lecture notes: Influence diagrams. http://ml.cs.tu-berlin.de/~mtoussai/notes/.

Toussaint, M., Charlin, L., and Poupart, P. (2008). Hierarchical POMDP controller optimization by likelihood maximization. In *Uncertainty in Artificial Intelligence (UAI 2008)*.

Toussaint, M., Harmeling, S., and Storkey, A. (2006). Probabilistic inference for solving (PO)MDPs. Technical Report EDI-INF-RR-0934, University of Edinburgh, School of Informatics.

Toussaint, M. and Storkey, A. (2006). Probabilistic inference for solving discrete and continuous state Markov Decision Processes. In *Proc. of the 23nd Int. Conf. on Machine Learning (ICML 2006)*, pages 945–952.

Verma, D. and Rao, R. P. N. (2006). Goal-based imitation as probabilistic inference over graphical models. In *Advances in Neural Information Processing Systems (NIPS 2005)*.

Vlassis, N. and Toussaint, M. (2009). Model-free reinforcement learning as mixture learning. In *Proc. of the 26rd Int. Conf. on Machine Learning (ICML 2009)*.

Zettlemoyer, L., Pasula, H., and Kaelbling, L. P. (2005). Learning planning rules in noisy stochastic worlds. In *Proc. of the Twentieth National Conf. on Artificial Intelligence (AAAI 05)*.