Efficient Reinforcement Learning in Factored MDPs

Michael Kearns
AT&T Labs
mkearns@research.att.com

Daphne Koller
Stanford University
koller@cs.stanford.edu

Abstract

We present a provably efficient and near-optimal algorithm for reinforcement learning in Markov decision processes (MDPs) whose transition model can be *factored* as a dynamic Bayesian network (DBN). Our algorithm generalizes the recent E^3 algorithm of Kearns and Singh, and assumes that we are given both an algorithm for approximate planning, and the graphical structure (but not the parameters) of the DBN. Unlike the original E^3 algorithm, our new algorithm exploits the DBN structure to achieve a running time that scales polynomially in the number of *parameters* of the DBN, which may be exponentially smaller than the number of global states.

1 Introduction

Kearns and Singh (1998) recently presented a new algorithm for reinforcement learning in Markov decision processes (MDPs). Their E^3 algorithm (for *Explicit Explore or Exploit*) achieves near-optimal performance in a running time and a number of actions which are polynomial in the number of states and a parameter T, which is the horizon time in the case of discounted return, and the mixing time of the optimal policy in the case of infinite-horizon average return. The E^3 algorithm makes no assumptions on the structure of the unknown MDP, and the resulting polynomial dependence on the number of states makes E^3 impractical in the case of very large MDPs. In particular, it cannot be easily applied to MDPs in which the transition probabilities are represented in the factored form of a *dynamic Bayesian network* (DBN). MDPs with very large state spaces, and such *DBN-MDPs* in particular, are becoming increasingly important as reinforcement learning methods are applied to problems of growing difficulty [Boutilier *et al.*, 1999].

In this paper, we extend the E^3 algorithm to the case of DBN-MDPs. The original E^3 algorithm relies on the ability to find optimal strategies in a *given* MDP — that is, to perform *planning*. This ability is readily provided by algorithms such as value iteration in the case of small state spaces. While the general planning problem is intractable in large MDPs, significant progress has been made recently on *approximate* solution algorithms for both DBN-MDPs in particular [Boutilier *et al.*, 1999], and for large state spaces in general [Kearns *et al.*, 1999; Koller and Parr, 1999]. Our new DBN-E^3 algorithm therefore assumes the existence of a procedure for finding approximately optimal policies in any *given* DBN-MDP. Our algorithm also assumes that the qualitative structure of the transition model is known, i.e., the underlying graphical structure of the DBN. This assumption is often reasonable, as the qualitative properties of a domain are often understood.

Using the planning procedure as a subroutine, DBN-E^3 explores the state space, learning the parameters it considers relevant. It achieves near-optimal performance in a running time and a number of actions that are polynomial in T and the *number of parameters in the DBN-MDP*, which in general is exponentially smaller than the number of global states. We further examine conditions under which the mixing time T of a policy in a DBN-MDP is polynomial in the number of parameters of the DBN-MDP. The "anytime" nature of DBN-E^3 allows it to compete with such policies in total running time that is bounded by a polynomial in the number of parameters.

2 Preliminaries

We begin by introducing some of the basic concepts of MDPs and factored MDPs. A *Markov Decision Process (MDP)* is defined as a tuple (S, A, R, P) where: S is a set of states; A is a set of actions; R is a *reward function* $R : S \mapsto [0, R_{max}]$, such that $R(s)$ represents the reward obtained by the agent in state s [1]; P is a *transition model* $P : S \times A \mapsto \Delta_S$, such that $P(s' \mid s, a)$ represents the probability of landing in state s' if the agent takes action a in state s.

Most simply, MDPs are described explicitly, by writing down a set of transition matrices and reward vectors — one for each action a. However, this approach is impractical for describing complex processes. Here, the set of states is typically described via a set of random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, where each X_i takes on values in some finite domain $\mathrm{Val}(X_i)$. In general, for a set of variables $\mathbf{Y} \subset \mathbf{X}$, an instantiation $\mathbf{y}$ assigns a value $x \in \mathrm{Val}(X)$ for every $X \in \mathbf{Y}$; we use $\mathrm{Val}(\mathbf{Y})$ to denote the set of possible instantiations to

[1] A reward function is sometimes associated with (state,action) pairs rather than with states. Our assumption that the reward depends only on the state is made purely to simplify the presentation; it has no effect on our results.

$\mathbf{Y}$. A state in this MDP is an assignment $\mathbf{x} \in \mathrm{Val}(\mathbf{X})$; the total number of states is therefore exponentially large in the number of variables. Thus, it is impractical to represent the transition model explicitly using transition matrices.

The framework of *dynamic Bayesian networks (DBNs)* allows us to describe a certain important class of such MDPs in a compact way. Processes whose state is described via a set of variables typically exhibit a weak form of decoupling — not all of the variables at time t directly influence the transition of a variable X_i from time t to time $t+1$. For example, in a simple robotics domain, the location of the robot at time $t+1$ may depend on its position, velocity, and orientation at time t, but not on what it is carrying, or on the amount of paper in the printer. DBNs are designed to represent such processes compactly.

Let $a \in A$ be an action. We first want to specify the transition model $P(\mathbf{x}' \mid \mathbf{x}, a)$. Let X_i denote the variable X_i at the current time and X_i' denote the variable at the next time step. The transition model for action a will consist of two parts — an underlying *transition graph* associated with a, and parameters associated with that graph. The transition graph is a 2-layer directed acyclic graph whose nodes are $\{X_1, \dots, X_n, X_1', \dots, X_n'\}$. All edges in this graph are directed from nodes in $\{X_1, \dots, X_n\}$ to nodes in $\{X_1', \dots, X_n'\}$; note that we are assuming that there are no edges between variables within a time slice. We denote the parents of X_i' in the graph by $\mathrm{Pa}_a(X_i')$. Intuitively, the transition graph for a specifies the *qualitative* nature of probabilistic dependencies in a single time step — namely, the new setting of X_i depends only on the current setting of the variables in $\mathrm{Pa}_a(X_i')$. To make this dependence *quantitative*, each node X_i' is associated with a *conditional probability table (CPT)* $P_a(X_i' \mid \mathrm{Pa}_a(X_i'))$. The transition probability $P(\mathbf{x}' \mid \mathbf{x}, a)$ is then defined to be $\prod_i P_a(x_i' \mid \mathbf{u}_i)$, where $\mathbf{u}_i$ is the setting in $\mathbf{x}$ of the variables in $\mathrm{Pa}_a(X_i')$.

We also need to provide a compact representation of the reward function. As in the transition model, explicitly specifying a reward for each of the exponentially many states is impractical. Again, we use the idea of factoring the representation of the reward function into a set of *localized* reward functions, each of which only depends on a small set of variables. In our robot example, our reward might be composed of several subrewards: for example, one associated with location (for getting too close to a wall), one associated with the printer status (for letting paper run out), and so on. More precisely, let $\mathcal{R}$ be a set of functions $R_1, \dots, R_k$; each function R_i is associated with a cluster of variables $\mathbf{C}_i \subset \{X_1, \dots, X_n\}$, such that R_i is a function from $\mathrm{Val}(\mathbf{C}_i)$ to $I\!\!R$. Abusing notation, we will use $R_i(\mathbf{x})$ to denote the value that R_i takes for the part of the state vector corresponding to $\mathbf{C}_i$. The reward function associated with the DBN-MDP at a state $\mathbf{x}$ is then defined to be $R(\mathbf{x}) = \sum_{i=1}^{k} R_i(\mathbf{x}) \in [0, R_{max}]$.

The following definitions for finite-length paths in MDPs will be of repeated technical use in the analysis. Let M be a Markov decision process, and let π be a policy in M. A *T-path in M* is a sequence p of $T+1$ states (that is, T transitions) of M: $p = \mathbf{x}_1, \dots, \mathbf{x}_T, \mathbf{x}_{T+1}$. The probability that p is traversed in M upon starting in state $\mathbf{x}_1$ and executing policy π is denoted $P_M^\pi[p] = \Pi_{k=1}^{T} P(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \pi(\mathbf{x}_k))$.

There are three standard notions of the expected *return* enjoyed by a policy in an MDP: the asymptotic discounted return, the asymptotic average return, and the finite-time average return. Like the original E^3 algorithm, our new generalization will apply to all three cases, and to convey the main ideas it suffices for the most part to concentrate on the finite-time average return. This is because our finite-time average return result can be applied to the asymptotic returns through either the *horizon time* $1/(1-\gamma)$ for the discounted case, or the *mixing time* of the optimal policy in the average case. (We examine the properties of mixing times in a DBN-MDP in Section 5.)

Let M be a Markov decision process, let π be a policy in M, and let p be a T-path in M. The *average return along* p in M is

$$U_M(p) = (1/T)(R(\mathbf{x}_1) + \cdots + R(\mathbf{x}_{T+1})).$$

The *T-step* (expected) average return from state $\mathbf{x}$ is $U_M^\pi(\mathbf{x}, T) = \sum_p P_M^\pi[p] U_M(p)$ where the sum is over all T-paths p in M that start at $\mathbf{x}$. Furthermore, we define the *optimal T-step average return* from $\mathbf{x}$ in M by $U_M^*(\mathbf{x}, T) = \max_\pi\{U_M^\pi(\mathbf{x}, T)\}$.

An important problem in MDPs is *planning*: finding the policy π^* that achieves optimal return in a *given* MDP. In our case, we are interested in achieving the optimal T-step average return. The complexity of all exact MDP planning algorithms depends polynomially on the number of states; this property renders all of these algorithms impractical for DBN-MDPs, where the number of states grows exponentially in the size of the representation. However, there has been recent progress on algorithms for *approximately* solving MDPs with large state spaces [Kearns *et al.*, 1999], particularly on ones represented in a factored way as an MDP [Boutilier *et al.*, 1999; Koller and Parr, 1999]. The focus of our work is on the reinforcement learning task, so we simply assume that we have access to a "black box" that performs approximate planning for a DBN-MDP.

Definition 2.1: A *μ-approximation T-step planning algorithm* for a DBN-MDP is one that, given a DBN-MDP M, produces a (compactly represented) policy π such that $U_M^\pi(\mathbf{x}, T) \geq (1-\mu) U_M^*(\mathbf{x}, T)$. ∎

We will charge our learning algorithm a single step of computation for each call to the assumed approximate planning algorithm. One way of thinking about our result is as a *reduction* of the problem of efficient learning in DBN-MDPs to the problem of efficient planning in DBN-MDPs.

Our goal is to perform model-based reinforcement learning. Thus, we wish to learn an approximate model from experience, and then exploit it (or explore it) by planning given the approximate model. In this paper, we focus on the problem of learning the model *parameters* (the CPTs), assuming that the model *structure* (the transition graphs) is given to us. It is therefore useful to consider the set of parameters that we wish to estimate. As we assumed that the rewards are deterministic, we can focus on the probabilistic parameters. (Our results easily extend to the case of stochastic rewards.) We define a *transition component* of the DBN-MDP to be a distribution $P_a(X_i' \mid \mathbf{u})$ for some action a and some particular instantiation $\mathbf{u}$ to the parents $\mathrm{Pa}_a(X_i')$ in the transition model. Note that the number of transition components is at most $\sum_{a,i} |\mathrm{Val}(\mathrm{Pa}_a(X_i'))|$, but may be much lower when a variable's behavior is identical for several actions.

3 Overview of the Original E^3

Since our algorithm for learning in DBN-MDPs will be a direct generalization of the E^3 algorithm of Kearns and Singh — hereafter abbreviated KS — we begin with an overview of that algorithm and its analysis. It is important to bear in mind that the original algorithm is designed only for the case where the total number of states N is small, and the algorithm runs in time polynomial in N.

E^3 is what is commonly referred to as an *indirect* or *model-based* algorithm: rather than maintaining only a current policy or value function, the algorithm maintains a model for the transition probabilities and the rewards for some *subset* of the states of the unknown MDP M. Although the algorithm maintains a partial model of M, it may choose to *never* build a *complete* model of M, if doing so is not necessary to achieve high return.

The algorithm starts off by doing *balanced wandering*: the algorithm, upon arriving in a state, takes the action it has tried the fewest times from that state (breaking ties randomly). At each state it visits, the algorithm maintains the obvious statistics: the reward received at that state, and for each action, the empirical distribution of next states reached (that is, the estimated transition probabilities).

A crucial notion is that of a *known state* — a state that the algorithm has visited "so many" times that the transition probabilities for that state are "very close" to their true values in M. This definition is carefully balanced so that "so many" times is still polynomially bounded, yet "very close" suffices to meet the simulation requirements below. An important observation is that we cannot do balanced wandering indefinitely before at least one state becomes known: by the Pigeonhole Principle, we will soon start to accumulate accurate statistics at some state.

The most important construction of the analysis is the *known-state MDP*. If $\mathcal{S}$ is the set of currently known states, the known-state MDP is simply an MDP $M_{\mathcal{S}}$ that is naturally *induced* on $\mathcal{S}$ by the full MDP M. Briefly, all transitions in M between states in $\mathcal{S}$ are preserved in $M_{\mathcal{S}}$, while all other transitions in M are "redirected" in $M_{\mathcal{S}}$ to lead to a single new, absorbing state that intuitively represents all of the unknown and unvisited states. Although E^3 does not have direct access to $M_{\mathcal{S}}$, by virtue of the definition of the known states, it does have a good *approximation* $\hat{M}_{\mathcal{S}}$.

The KS analysis hinges on two central technical lemmas. The first is called the Simulation Lemma, and it establishes that $\hat{M}_{\mathcal{S}}$ has good *simulation accuracy*: that is, the expected T-step return of any policy in $\hat{M}_{\mathcal{S}}$ is close to its expected T-step return in $M_{\mathcal{S}}$. Thus, at any time, $\hat{M}_{\mathcal{S}}$ is a useful *partial* model of M, for that part of M that the algorithm "knows" very well.

The second central technical lemma is the "Explore or Exploit" Lemma. It states that either the optimal (T-step) policy in M achieves its high return by staying (with high probability) in the set $\mathcal{S}$ of currently known states, or the optimal policy has significant probability of *leaving* $\mathcal{S}$ within T steps. Most importantly, the algorithm can detect which of these two is the case; in the first case, it can simulate the behavior of the optimal policy by finding a high-return *exploitation* policy in the partial model $\hat{M}_{\mathcal{S}}$, and in the second case, it can replicate the behavior of the optimal policy by finding an *exploration* policy that quickly reaches the additional absorbing state of the partial model $\hat{M}_{\mathcal{S}}$. Thus, by performing two off-line planning computations on $\hat{M}_{\mathcal{S}}$, the algorithm is guaranteed to find either a way to get near-optimal return for the next T steps, or a way to improve the statistics at an unknown or unvisited state within the next T steps. KS show that this algorithm ensures near-optimal return in time polynomial in N.

4 The DBN-E^3 Algorithm

Our goal is to derive a generalization of E^3 for DBN-MDPs, and to prove for it a result analogous to that of KS — but with a polynomial dependence not on the number of states N, but on the number of CPT parameters ℓ in the DBN model. Our analysis closely mirrors the original, but requires a significant generalization of the Simulation Lemma that exploits the structure of a DBN-MDP, a modified construction of $\hat{M}_{\mathcal{S}}$ that can be represented as a DBN-MDP, and a number of alterations of the details.

Like the original E^3 algorithm, DBN-E^3 will build a model of the unknown DBN-MDP on the basis of its experience, but now the model will be represented in a compact, factorized form. More precisely, suppose that our algorithm is in state $\mathbf{x}$, executes action a, and arrives in state $\mathbf{x}'$. This experience will be used to update all the appropriate CPT entries of our model — namely, all the estimates $\hat{P}_a(x_i' \mid \mathbf{u}_i)$ are updated in the obvious way, where as usual $\mathbf{u}_i$ is the setting of $\mathrm{Pa}_a(X_i')$ in $\mathbf{x}$. We will also maintain counts $C_a(x_i', \mathbf{u}_i)$ of the number of times $\hat{P}_a(x_i' \mid \mathbf{u}_i)$ has been updated.

Recall that a crucial element of the original E^3 analysis was the notion of a *known state*. In the original analysis, it was observed that if N is the total number of states, then after $O(N)$ experiences some state must become known by the Pigeonhole Principle. We cannot hope to use the same logic here, as we are now in a DBN-MDP with an exponentially large number of states. Rather, we must "pigeonhole" not on the number of states, but on the number of parameters required to specify the DBN-MDP. Towards this goal, we will say that the *CPT entry* $\hat{P}_a(x_i' \mid \mathbf{u}_i)$ is *known* if it has been visited "enough" times to ensure that, with high probability

$$|P_a(x_i' \mid \mathbf{u}_i) - \hat{P}_a(x_i' \mid \mathbf{u}_i)| \leq \alpha.$$

We now would like to establish that if, for an appropriate choice of α, *all* CPT entries are known, then our approximate DBN-MDP can be used to accurately estimate the expected return of any policy in the true DBN-MDP. This is the desired generalization of the original Simulation Lemma. As in the original analysis, we will eventually apply it to a generalization of the induced MDP $M_{\mathcal{S}}$, in which we deliberately restrict attention to only the known CPT entries.

4.1 The DBN-MDP Simulation Lemma

Let M and $\hat{M}$ be two DBN-MDPs over the same state space with the same transition graphs for every action a, and with the same reward functions. Then we say that $\hat{M}$ is an α-*approximation* of M if for every action a and node X_i' in the transition graphs, for every setting $\mathbf{u}$ of $\mathrm{Pa}_a(X_i')$, and for every possible value x_i' of X_i',

$$|P_a(x_i' \mid \mathbf{u}) - \hat{P}_a(x_i' \mid \mathbf{u})| \leq \alpha$$

where $P_a(\cdot \mid \cdot)$ and $\hat{P}_a(\cdot \mid \cdot)$ are the CPTs of M and $\hat{M}$, respectively.

Lemma 4.1: *Let M be any DBN-MDP over n state variables with ℓ CPT entries in the transition model, and let $\hat{M}$ be an α-approximation of M, where $\alpha = O((\epsilon/(T^2\ell R_{max}))^2)$. Then for any policy π, and for any state $\mathbf{x}$, $|U_M^\pi(\mathbf{x},T) - U_{\hat{M}}^\pi(\mathbf{x},T)| \leq \epsilon$.*

Proof: (Sketch) Let us fix a policy π and state $\mathbf{x}$. Recall that for any next state $\mathbf{x}'$ and any action a, the transition probability factorizes via the CPTs as $P(\mathbf{x}' \mid \mathbf{x}, a) = \prod_i P_a(x_i' \mid \mathbf{u}_i)$. where $\mathbf{u}_i$ is the setting of $\mathrm{Pa}_a(X_i')$ in $\mathbf{x}$. Let us say that $P(\mathbf{x}' \mid \mathbf{x}, a)$ *contains a β-small factor* if any of its CPT factors $P_a(x_i' \mid \mathbf{u}_i)$ is smaller than β. Note that a transition probability may actually be quite small itself (exponentially small in n) without necessarily containing a β-small factor.

Our first goal is to show that trajectories in M and $\hat{M}$ that cross transitions containing a β-small CPT factor can be "thrown away" without much error. Consider a random trajectory of T steps in M from state $\mathbf{x}$ following policy π. It can be shown that the probability that such a trajectory will cross at least one transition $P(\mathbf{x}' \mid \mathbf{x}, a)$ that contains a β-small factor is at most $T\ell\beta$. Essentially, the probability that at any step, any particular β-small transition (CPT factor) will be taken by any particular variable X_i is at most β. A simple union argument over the CPT entries and the T time steps gives the desired bound. Therefore, the total contribution to the difference $|U_M^\pi(\mathbf{x},T) - U_{\hat{M}}^\pi(\mathbf{x},T)|$ by these trajectories can be shown to be at most $T^2 R_{max}\ell(\alpha+\beta)$. We will thus ignore such trajectories for now.

The key advantage of eliminating β-small factors is that we can convert additive approximation guarantees into multiplicative ones. Let p be any path of length T. If all the relevant CPT factors are greater than β, and we let $\Delta = \alpha/\beta$, it can be shown that

$$(1-\Delta)^{Tn} P_M^\pi[p] \leq \hat{P}_M^\pi[p] \leq (1+\Delta)^{Tn} P_M^\pi[p].$$

In other words, ignoring β-small CPT factors, the distributions on paths induced by π in M and $\hat{M}$ are quite similar. From this it follows that, for the upper bound,[2]

$$U_{\hat{M}}^\pi(\mathbf{x},T) \leq (1+\Delta)^{Tn} U_M^\pi(\mathbf{x},T) + T^2 R_{max}\ell(\alpha+2\beta).$$

For the choices $\beta = \sqrt{\alpha}$, $\alpha = O((\epsilon/(T^2\ell R_{max}))^2)$ the lemma is obtained. ∎

[2]The lower bound argument is entirely symmetric.

Returning to the main development, we can now give a precise definition of a known CPT entry. It is a simple application of Chernoff bounds to show that provided the count $C_a(x_i', \mathbf{u}_i)$ exceeds $O(1/\alpha^2 \log(1/\delta))$, $\hat{P}_a(x_i' \mid \mathbf{u}_i)$ has additive error at most α with probability at least $1-\delta$. We thus say that this CPT entry is known if its count exceeds the given bound for the choice $\alpha = O((\epsilon/(T^2 nvR_{max}))^2)$ specified by the DBN-MDP Simulation Lemma. The DBN-MDP Simulation Lemma shows that if *all* CPT entries are known, then our approximate model $\hat{M}$ can be used to find a near-optimal policy in the true DBN-MDP M.

Note that we can *identify* which CPT entries are known via the counts $C_a(x_i', \mathbf{u}_i)$. Thus, if we are at a state $\mathbf{x}$ for which at least one of the associated CPT entries $\hat{P}_a(x_i' \mid \mathbf{u}_i)$ is unknown, by taking action a we then obtain an experience that will increase the corresponding count $C_a(x_i', \mathbf{u}_i)$. Thus, in analogy with the original E^3, as long as we are encountering unknown CPT entries, we can continue taking actions that increase the quality of our model — but now rather than increasing counts on a per-state basis, the DBN-MDP Simulation Lemma shows why it suffices to increase the counts on a per-CPT entry basis, which is crucial for obtaining the running time we desire. We can thus show that if we encounter unknown CPT entries for a number of steps that is polynomial in the total number ℓ of *CPT entries* and $1/\epsilon$, there can no longer be any unknown CPT entries, and we know the true DBN-MDP well enough to solve for a near-optimal policy.

However, similar to the original algorithm, the real difficulty arises when we are in a state with no unknown CPT entries, yet there do remain unknown CPT entries elsewhere. Then we have no guarantee that we can improve our model at the next step. In the original algorithm, this was solved by defining the known-state MDP $M_{\mathcal{S}}$, and proving the aforementioned "Explore or Exploit" Lemma. Duplicating this step for DBN-MDPs will require another new idea.

4.2 The DBN-MDP "Explore or Exploit" Lemma

In our context, when we construct a known-state MDP, we must satisfy the additional requirement that the known-state MDP preserve the DBN structure of the original problem, so that if we have a planning algorithm for DBN-MDPs that exploits the structure, we can then apply it to the known-state MDP[3]. Therefore, we cannot just introduce a new "sink state" to represent that part of M that is unknown to us; we must also show how this "sink state" can be represented as a setting of the state variables of a DBN-MDP.

We present a new construction, which extends the idea of "known states" to the idea of "known transitions". We say that a transition component $P_a(X_i' \mid \mathbf{u})$ is *known* if all of its CPT entries are known. The basic idea is that, while it is impossible to check locally whether a state is known, it is easy to check locally whether a transition component is known.

Let $\mathcal{T}$ be the set of known transition components. We define the known-transition DBN-MDP $M_{\mathcal{T}}$ as follows. The

[3]Certain approaches to approximate planning in large MDPs do not require any structural assumptions [Kearns *et al.*, 1999], but we anticipate that the most effective DBN-MDP planning algorithms eventually will.

model behaves identically to M as long as only known transitions are taken. As soon as an unknown transition is taken for some variable X'_i, the variable X'_i takes on a new *wandering* value w, which we introduce into the model. The transition model is defined so that, once a variable takes on the value w, its value never changes. The reward function is defined so that, once at least one variable takes on the wandering value, the total reward is nonpositive. These two properties give us the same overall behavior that KS got by making a sink state for the set of unknown states.

Definition 4.2: Let M be a DBN-MDP and let $\mathcal{T}$ be any subset of the transition components in the model. The *induced DBN-MDP on* $\mathcal{T}$, denoted $M_{\mathcal{T}}$, is defined as follows:

- $M_{\mathcal{T}}$ has the same set of state variables as M; however, in $M_{\mathcal{T}}$, each variable X_i has, in addition to its original set of values $\mathrm{Val}^M(X_i)$, a new value w.
- $M_{\mathcal{T}}$ has the same transition graphs as M. For each a, i, and $\mathbf{u} \in \mathrm{Val}^M(\mathrm{Pa}_a(X'_i))$, we have that $P_a^{M_{\mathcal{T}}}(X'_i \mid \mathbf{u}) = P_a^M(X'_i \mid \mathbf{u})$ if the corresponding transition component is in $\mathcal{T}$; in all other cases, $P_a^{M_{\mathcal{T}}}(w \mid \mathbf{u}) = 1$, and $P_a^{M_{\mathcal{T}}}(x_i \mid \mathbf{u}) = 0$ for all $x_i \in \mathrm{Val}^M(X_i)$.
- $M_{\mathcal{T}}$ has the same set $\mathcal{R}$ as M. For each $i = 1, \ldots, k$ and $\mathbf{c} \in \mathrm{Val}^M(\mathbf{C}_i)$, we have that $R_i^{M_{\mathcal{T}}}(\mathbf{c}) = R_i^M(\mathbf{c})$. For other vectors $\mathbf{c}$, we have that $R_i^{M_{\mathcal{T}}}(\mathbf{c}) = -R_{max}$. ∎

With this definition, we can prove the analogue to the "Explore or Exploit" Lemma (details omitted).

Lemma 4.3:*Let M be any DBN-MDP, let $\mathcal{T}$ be any subset of the transition components of M, and let $M_{\mathcal{T}}$ be the induced MDP on M. For any $\mathbf{x} \in S$, any T, and any $1 > \tau > 0$, either there exists a policy π in $M_{\mathcal{T}}$ such that $U^{\pi}_{M_{\mathcal{T}}}(\mathbf{x}, T) \geq U^*_M(\mathbf{x}, T) - \tau$, or there exists a policy π in $M_{\mathcal{T}}$ such that the probability that a walk of T steps following π will take at least one transition not in $\mathcal{T}$ exceeds $\tau/((k+1)TR_{max})$.*

This lemma essentially asserts that either there exists a policy that already achieves near-optimal (global) return by staying only in the local model $M_{\mathcal{T}}$, or there exists a policy that quickly exits the local model.

4.3 Putting It All Together

We now have all the pieces to finish the description and analysis of the DBN-E^3 algorithm. The algorithm initially executes balanced wandering for some period of time. After some number of steps, by the Pigeonhole Principle one or more transition components become known. When the algorithm reaches a known state $\mathbf{x}$ — one where all the transition components are known — it can no longer perform balanced wandering. At that point, the algorithm performs approximate off-line policy computations for two different DBN-MDPs. The first corresponds to attempted exploitation, and the second to attempted exploration.

Let $\mathcal{T}$ be the set of known transitions at this step. In the attempted exploitation computation, the DBN-E^3 algorithm would like to find the optimal policy on the induced DBN-MDP $M_{\mathcal{T}}$. Clearly, this DBN-MDP is not known to the algorithm. Thus, we use its approximation $\hat{M}_{\mathcal{T}}$, where the true transition probabilities are replaced with their current approximation in the model. The definition of $M_{\mathcal{T}}$ uses only the CPT entries of known transition components. The Simulation Lemma now tells us that, for an appropriate choice of α — a choice that will result in a definition of known transition that requires the corresponding count to be only polynomial in $1/\epsilon$, n, v, and T — the return of any policy π in $\hat{M}_{\mathcal{T}}$ is within ϵ of its return in $M_{\mathcal{T}}$. We will specify a choice for ϵ later (which in turn sets the choice of α and the definition of known state).

Let us now consider the two cases in the "Explore or Exploit" Lemma. In the exploitation case, there exists a policy π in $M_{\mathcal{T}}$ such that $U^{\pi}_{M_{\mathcal{T}}}(\mathbf{x}, T) \geq U^*_M(\mathbf{x}, T) - \tau$. (Again, we will discuss the choice of τ below.) From the Simulation Lemma, we have that $U^{\pi}_{\hat{M}_{\mathcal{T}}}(\mathbf{x}, T) \geq U^*_M(\mathbf{x}, T) - (\tau + \epsilon)$. Our approximate planning algorithm returns a policy π' whose value in $\hat{M}_{\mathcal{T}}$ is guaranteed to be a multiplicative factor of at most $1-\mu$ away from the optimal policy in $\hat{M}_{\mathcal{T}}$. Thus, we are guaranteed that $U^{\pi'}_{\hat{M}_{\mathcal{T}}}(\mathbf{x}, T) \geq (1-\mu)(U^*_M(\mathbf{x}, T) - (\tau+\epsilon))$. Therefore, in the exploitation case, our approximate planner is guaranteed to return a policy whose value is close to the optimal value.

In the exploration case, there exists a policy π in $M_{\mathcal{T}}$ (and therefore in $\hat{M}_{\mathcal{T}}$) that is guaranteed to take an unknown transition within T steps with some minimum probability. Our goal now is to use our approximate planner to find such a policy. In order to do that, we need use a slightly different construction $M'_{\mathcal{T}}$ ($\hat{M}'_{\mathcal{T}}$). The transition structure of $M'_{\mathcal{T}}$ is identical to that of $M_{\mathcal{T}}$. However, the rewards are now different. Here, for each $i = 1, \ldots, k$ and $\mathbf{c} \in \mathrm{Val}^M(\mathbf{C}_i)$, we have that $R_i^{M'_{\mathcal{T}}}(\mathbf{c}) = 0$; for other vectors $\mathbf{c}$, we have that $R_i^{M'_{\mathcal{T}}}(\mathbf{c}) = 1$. Now let π' be the policy returned by our approximate planner on the DBN-MDP $\hat{M}'_{\mathcal{T}}$. It can be shown that the probability that a T-step walk following π' will take at least one unknown transition is at least $(1-\mu)(\tau/((k+1)TR_{max}) - \epsilon)/kT$.

To summarize: our approximate planner either finds an exploitation policy π in $\hat{M}_{\mathcal{T}}$ that enjoys actual return $U^{\pi}_M(\mathbf{x}, T) \geq (1-\mu)(U^*_M(\mathbf{x}, T) - (\tau+\epsilon))$ from our current state $\mathbf{x}$, or it finds an exploitation policy in $\hat{M}'_{\mathcal{T}}$ that has probability at least $p = (1-\mu)(\tau/((k+1)TR_{max}) - \epsilon)/kT$ of improving our statistics at an unknown transition in the next T steps. Appropriate choices for ϵ and τ yield our main theorem, which we are now finally ready to describe.

Recall that for expository purposes we have concentrated on the case of T-step average return. However, as for the original E^3, our main result can be stated in terms of the asymptotic discounted and average return cases. We omit the details of this translation, but it is a simple matter of arguing that it suffices to set T to be either $(1/(1-\gamma))\log(1/\epsilon)$ (discounted) or the mixing time of the optimal policy (average).

Theorem 4.4: *(Main Theorem) Let M be a DBN-MDP with ℓ total entries in the CPTs.*

- *(Undiscounted case) Let T be the mixing time of the policy achieving the optimal average asymptotic return U^* in M. There exists an algorithm DBN-E^3 that, given access to a μ-approximation planning algorithm for DBN-*

MDPs, and given inputs $\epsilon, \delta, \ell, T$ and U^*, takes a number of actions and computation time bounded by a polynomial in $1/(1-\mu), 1/\epsilon, 1/\delta, \ell, T$, and R_{max}, and with probability at least $1-\delta$, achieves total actual return exceeding $U^* - \epsilon$.

- *(Discounted case) Let V^* denote the value function for the policy with the optimal expected discounted return in M. There exists an algorithm DBN-E^3 that, given access to a μ-approximation planning algorithm for DBN-MDPs, and given inputs ϵ, δ, ℓ and V^*, takes a number of actions and computation time bounded by a polynomial in $1/(1-\mu), 1/\epsilon, 1/\delta, \ell$, the horizon time $T = 1/(1-\gamma)$, and R_{max}, and with probability at least $1-\delta$, will halt in a state $\mathbf{x}$, and output a policy $\hat{\pi}$, such that $V_M^{\hat{\pi}}(\mathbf{x}) \geq V^*(\mathbf{x}) - \epsilon$.*

Some remarks:

- The loss in policy quality induced by the approximate planning subroutine translates into degradation in the running time of our algorithm.
- As with the original E^3, we can eliminate knowledge of the optimal returns in both cases via search techniques.
- Although we have stated our asymptotic undiscounted average return result in terms of the mixing time of the optimal policy, we can instead give an "anytime" algorithm that "competes" against policies with longer and longer mixing times the longer it is run. (We omit details, but the analysis is analogous to the original E^3 analysis.) This extension is especially important in light of the results of the following section, where we examine properties of mixing times in DBN-MDPs.

5 Mixing Time Bounds for DBN-MDPs

As in the original E^3 paper, our average case result depends on the amount of time T that it takes the target policy to mix. This dependence is unavoidable. If some of the probabilities are very small, so that the optimal policy cannot easily reach the high-reward parts of the space, it is unrealistic to expect the reinforcement learning algorithm to do any better.

In the context of a DBN-MDP, however, this dependence is more troubling. The size of the state space is exponentially large, and virtually all of the probabilities for transitioning from one state to the next will be exponentially small (because a transition probability is the product of n numbers that are ≤ 1). Indeed, one can construct very reasonable DBN-MDPs that have an exponentially long mixing time. For example, a DBN representing the Markov chain of an Ising model [Jerrum and Sinclair, 1993] has small parent sets (at most four parents per node), and CPT entries that are reasonably large. Nevertheless, the mixing time of such a DBN can be exponentially large in n.

Given that even "reasonable" DBNs such as this can have exponential mixing times, one might think that this is the typical situation — that is, that most DBN-MDPs have an exponentially long mixing time, reintroducing the exponential dependence on n that we have been trying so hard to avoid. We now show that this is not always the case. We provide a tool for analyzing the mixing time of a policy in a DBN-MDP, which can give us much better bounds on the mixing time. In particular, we demonstrate a class of DBN-MDPs and associated policies for which we can guarantee rapid mixing.

Note that any fixed policy in a DBN-MDP defines a Markov chain whose transition model is represented as a DBN. We therefore begin by considering the mixing time of a pure DBN, with no actions. We then extend that analysis to the mixing rate for a fixed policy in a DBN-MDP.

Definition 5.1: Let Q be a transition model for a Markov chain, and let $\{X^{(t)}\}_{t=1}^{\infty}$ represent the state of the chain. Let $S = \{x_1, \ldots, x_s\}$. Let μ_j be the stationary probability of x_j in this Markov chain. We say that the Markov chain Q is ϵ-mixed at time m if $\max_{i,j} |P(X^{(t)} = x_j \mid X^{(1)} = x_i) - \mu_j| \leq \epsilon$. ∎

Our bounds on mixing times make use of the *coupling method* [Lindvall, 1992]. The idea of the coupling method is as follows: we run two copies of the Markov chain in parallel, from different starting points. Our goal is to make the states of the two processes coalesce. Intuitively, the first time the states of the two copies are the same, the initial states have been "forgotten", which corresponds to the processes having mixed.

More precisely, consider a transition matrix Q over some state space S. Let Q^* be a transition matrix over the state space $S \times S$, such that if $\{(Y^{(t)}, Z^{(t)})\}_{t=1}^{\infty}$ is the Markov chain for Q^*, then the separated Markov chains $\{Y^{(t)}\}_{t=1}^{\infty}$ and $\{Z^{(t)}\}_{t=1}^{\infty}$ both evolve according to Q. Let τ be the random variable that represents the *coupling time* — the smallest m for which $Y^{(m)} = Z^{(m)}$. The following lemma establishes the correspondence between mixing and coupling times.

Lemma 5.2: *For any ϵ, let m be such that for any $i, j = 1, \ldots, s$, $P(\tau > m \mid Y^{(1)} = x_i, Z^{(1)} = x_j) \leq \epsilon$. Then Q is ϵ-mixed at time m.*

Thus, to show that a Markov chain is ϵ-mixed by some time m, we need only construct a coupled chain and show that the probability that this chain has not coupled by time m decreases very rapidly in m.

The coupling method allows us to construct the joint chain over $(Y^{(t)}, Z^{(t)})$ in any way that we want, as long as each of the two chains in isolation has the same dynamics as the original Markov chain Q. In particular, we can *correlate* the transitions of the two processes, so as to make their states coincide faster than they would if each was picked independently of the other. That is, we choose $Y^{(t+1)}$ and $Z^{(t+1)}$ to be equal to each other whenever possible, subject to the constraints on the transition probabilities. More precisely, let $Y^{(t)} = x_i$ and $Z^{(t)} = x_j$. For any value $x \in S$, we can make the event $Y^{(t+1)} = x_i, Z^{(t+1)} = x_j$ have a probability that is the smaller of $P(X' = x_k \mid X = x_i)$ and $P(X' = x_k \mid X = x_j)$. Compare this to the probability of this event if the two processes were independent, which is the product of these two numbers rather than their minimum. Overall, by correlating the two processes as much as possible, and considering the worst case over the current state of the process, we can guarantee that, at every step, the two processes couple with probability at least

$$\min_{i,j} \sum_k \min[P(X' = x_k \mid X = x_i), P(X' = x_k \mid X = x_j)]$$

This quantity represents the amount of probability mass that any two transition distributions are guaranteed to have in common. It is called the *Dobrushin coefficient*, and is the contraction rate for L_1-norm [Dobrushin, 1956] in Markov chains.

Now, consider a DBN over the state variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. As above, we create two copies of the process, letting $Y_1, \ldots, Y_n$ denote the variables in the first component of the coupled Markov chain, and $Z_1, \ldots, Z_n$ denote those in the second component. Our goal is to construct a Markov chain over $\mathbf{Y}, \mathbf{Z}$ such that both $\mathbf{Y}$ and $\mathbf{Z}$ separately have the same dynamics as $\mathbf{X}$ in the original DBN.

Our construction of the joint Markov chain is very similar to the one used above, except that will now choose the transition of each *variable pair* Y_i and Z_i so as to maximize the probability that they couple (assume the same value). As above, we can guarantee that Y_i and Z_i couple at any time t with probability at least

$$\beta_i = \min_{\mathbf{u},\mathbf{u}' \in \mathrm{Val}\left(\mathrm{Pa}(X'_i)\right)} \left\{ \sum_{x_i \in \mathrm{Val}(X_i)} \min[P(x_i \mid \mathbf{u}), P(x_i \mid \mathbf{u}')] \right\}$$

This coefficient was defined by [Boyen and Koller, 1998] in their analysis of the contraction rate of DBNs. Note that β_i depends only on the numbers in a *single* CPT of the DBN. Assuming that the transition probabilities in each CPT are not too extreme, the probability that any single variable couples will be reasonably high.

Unfortunately, this bound is not enough to show that all of the variable pairs couple within a short time. The problem is that it is not enough for two variables $Y_i^{(t)}$ and $Z_i^{(t)}$ to couple, as process dynamics may force us to decouple them at subsequent time slices. To understand this issue, consider a simple process with two variables X_1, X_2, and a transition graph with the edges $X_1 \to X'_1$, $X_2 \to X'_2$, $X_1 \to X'_2$. Assume that at time t, the variable pair $Y_2^{(t)}, Z_2^{(t)}$ has coupled with value x_2, but $Y_1^{(t)}, Z_1^{(t)}$ has not, so that $Y_1^{(t)} = x_1$ and $Z_1^{(t)} = x'_1$. At the next time slice, we must select $Y_2^{(t+1)}, Z_2^{(t+1)}$ from two different distributions — $P(X'_2 \mid x_1, x_2)$ and $P(X'_2 \mid x'_1, x_2)$, respectively. Thus, our sampling process may be forced to give them different values, decoupling them again.

As this example clearly illustrates, it is not enough for a variable pair to couple momentarily. In order to eventually couple the two processes as a whole, we need to make each variable pair a *stable pair* — i.e., we need to guarantee that our sampling process can keep them coupled from then on. In our example, the pair Y_1, Z_1 is stable as soon as it first couples. And once Y_1, Z_1 is stable, then Y_2, Z_2 will also be stable as soon as it couples. However, if Y_2, Z_2 couples while Y_1, Z_1 is not yet stable, then the sampling process cannot guarantee stability.

In general, a variable pair can only be stable if their parents are also stable. So what happens if we add the edge $X_2 \to X'_1$ to our transition model? In this case, neither Y_1, Z_1 nor Y_2, Z_2 can stabilize in isolation. They can only stabilize if they couple simultaneously.

This discussion leads to the following definition.

Definition 5.3: Consider a DBN over the state variables $X_1, \ldots, X_n$. The *dependency graph* $\mathcal{D}$ for the DBN is a directed cyclic graph whose nodes are $X_1, \ldots, X_n$ and where there is a directed edge from X_i to X_j if there is an edge in the transition graph of the DBN from X_i to X'_j. ▮

Hence, there is a directed path from X_i to X_j in $\mathcal{D}$ iff $X_i^{(t)}$ influences $X_j^{(t')}$ for some $t' > t$. We assume that the transition graph of the DBN always has arcs $X_i \to X'_i$, so that the every node in $\mathcal{D}$ has a self-loop.

Let $\Gamma_1, \ldots, \Gamma_l$ be the maximal strongly connected components in $\mathcal{D}$, sorted so that if $i < j$, there are no directed edges from Γ_j to Γ_i. Our analysis will be based on stabilizing the Γ_i's in succession. (We note that we provide only a rough bound; a more refined analysis is possible.) Let $\beta = \min_i \beta_i$ and $g = \max_j |\Gamma_j|$. Assume that $\Gamma_1, \ldots, \Gamma_{i-1}$ have all stabilized by time t. In order for Γ_i to stabilize, all of the variables need to couple at exactly the same time. This event happens at time t with probability $\geq \beta^g$. As soon as Γ_i stabilizes, we can move on to stabilizing Γ_{i+1}. When all the Γ_i's have stabilized, we are done.

Theorem 5.4: *For any $\epsilon \geq 0$, the Markov chain corresponding to a DBN as described above is ϵ-mixed at time m provided*

$$m \geq \frac{8l}{\beta^g} \log(1/\epsilon).$$

Thus, the mixing time of a DBN grows exponentially with the size of the largest component in the dependency graph, which may be significantly smaller than the total number of variables in a DBN. Indeed, in two real-life DBNs — BAT [Forbes *et al.*, 1995] with ten state variables, and WATER [Jensen *et al.*, 1989] with eight — the maximal cluster size is 3–4.

It remains only to extend this analysis to DBN-MDPs, where we have a policy π. Our stochastic coupling scheme must now deal with the fact that the actions taken at time t in the two copies of the process may be different. The difficulty is that different actions at time t correspond to different transition models. If a variable X_i has a different transition model in different transition graphs P_a, it will use a different transition distribution if the action is not the same. Hence X_i cannot stabilize until we are guaranteed that the same action is taken in both copies. That is, the action must also stabilize. The action is only guaranteed to have stabilized when all of the variables on which the choice of action can possibly depend have stabilized. Otherwise, we might encounter a pair of states in which we are forced to use different actions in the two copies.

We can analyze this behavior by extending the dependency graph to include a new node corresponding to the choice of action. We then see what assumptions allow us to bound the set of incoming and outgoing edges. We can then use the same analysis described above to bound the mixing time. The outgoing edges correspond to the effect of an action. In many processes, the action only directly affects the transition model of a small number of state variables in the process. In other words, for many variables X_i, we have that $\mathrm{Pa}_a(X_i)$ and $P_a(X_i \mid \mathrm{Pa}_a(X_i))$ are the same for all a. In this case, the new action node will only have outgoing edges to the remaining variables (those for which the transition model might differ). We note that such localized influence models have a long history both for *influence diagram* [Howard and Matheson, 1984] and for DBN-MDPs [Boutilier *et al.*, 1999].

Now, consider outgoing edges. In general, the optimal policy might well be such that the action depends on every variable. However, the mere representation of such a policy may be very complex, rendering its use impractical in a DBN-MDP with many variables. Therefore, we often want to restrict attention to a simpler class of policies, such as a small finite state machine or a small decision tree. If our target policy is such that the choice of action only depends on a small number of variables, then there will only be a small number of incoming edges into the action node in the dependency graph.

Having integrated the action node into the dependency graph, our analysis above holds unchanged. The only difference from a random variable is that we do not have to include the action node when computing the size of the Γ_i that contains it, as we do not have to stochastically make it couple; rather, it couples immediately once its parents have coupled.

Finally, we note that this analysis easily accommodates DBN-MDPs where the decision about the action is also decomposed into several independent decisions (e.g., as in [Meuleau *et al.*, 1998]). Different component decisions can influence different subsets of variables, and the choice of action in each one can depend on different subsets of variables. Each decision forms a separate node in the dependency graph, and can stabilize independently of the other decisions.

The analysis above gives us techniques for estimating the mixing rate of policies in DBN-MDPs. In particular, if we want to focus on getting a good steady-state return from DBN-E^3 *in a reasonable amount of time*, this analysis shows us how to restrict attention to policies that are guaranteed to mix rapidly given the structure of the given DBN-MDP.

6 Conclusions

Structured probabilistic models, and particularly Bayesian networks, have revolutionized the field of reasoning under uncertainty by allowing compact representations of complex domains. Their success is built on the fact that this structure can be exploited effectively by inference and learning algorithms. This success leads one to hope that similar structure can be exploited in the context of planning and reinforcement learning under uncertainty. This paper, together with the recent work on representing and reasoning with factored MDPs [Boutilier *et al.*, 1999], demonstrate that substantial computational gains can indeed be obtained from these compact, structured representations.

This paper leaves many interesting problems unaddressed. Of these, the most intriguing one is to allow the algorithm to learn the model structure as well as the parameters. The recent body of work on learning Bayesian networks from data [Heckerman, 1995] lays much of the foundation, but the integration of these ideas with the problems of exploration/exploitation is far from trivial.

Acknowledgements

We are grateful to the members of the DAGS group for useful discussions, and particularly to Brian Milch for pointing out a problem in an earlier version of this paper. The work of Daphne Koller was supported by the ARO under the MURI program "Integrated Approach to Intelligent Systems," by ONR contract N66001-97-C-8554 under DARPA's HPKB program, and by the generosity of the Powell Foundation and the Sloan Foundation.

References

[Boutilier *et al.*, 1999] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 1999. To appear.

[Boyen and Koller, 1998] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. UAI*, pages 33–42, 1998.

[Dobrushin, 1956] R.L. Dobrushin. Central limit theorem for nonstationary Markov chains. *Theory of Probability and its Applications*, pages 65–80, 1956.

[Forbes *et al.*, 1995] J. Forbes, T. Huang, K. Kanazawa, and S.J. Russell. The BATmobile: Towards a Bayesian automated taxi. In *Proc. IJCAI*, 1995.

[Heckerman, 1995] D. Heckerman. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1995.

[Howard and Matheson, 1984] R. A. Howard and J. E. Matheson. Influence diagrams. In R. A. Howard and J. E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, pages 721–762. Strategic Decisions Group, Menlo Park, California, 1984.

[Jensen *et al.*, 1989] F.V. Jensen, U. Kjærulff, K.G. Olesen, and J. Pedersen. An expert system for control of waste water treatment—a pilot project. Technical report, Judex Datasystemer A/S, Aalborg, 1989. In Danish.

[Jerrum and Sinclair, 1993] M. Jerrum and A. Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22:1087–1116, 1993.

[Kearns and Singh, 1998] M. Kearns and S.P. Singh. Near-optimal performance for reinforcement learning in polynomial time. In *Proc. ICML*, pages 260–268, 1998.

[Kearns *et al.*, 1999] M. Kearns, Y. Mansour, and A. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *these proceedings*, 1999.

[Koller and Parr, 1999] D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs. In *these proceedings*, 1999.

[Lindvall, 1992] T. Lindvall. *Lectures on the Coupling Method*. Wiley, 1992.

[Meuleau *et al.*, 1998] N. Meuleau, M. Hauskrecht, K-E. Kim, L. Peshkin, L.P. Kaelbling, T. Dean, and C. Boutilier. Solving very large weakly coupled Markov decision processes. In *Proc. AAAI*, pages 165–172, 1998.