Feature Dynamic Bayesian Networks

Marcus Hutter
RSISE @ ANU and SML @ NICTA
Canberra, ACT, 0200, Australia
marcus@hutter1.net www.hutter1.net

Abstract

Feature Markov Decision Processes (ΦMDPs) [Hut09] are well-suited for learning agents in general environments. Nevertheless, unstructured (Φ)MDPs are limited to relatively simple environments. Structured MDPs like Dynamic Bayesian Networks (DBNs) are used for large-scale real-world problems. In this article I extend ΦMDP to ΦDBN. The primary contribution is to derive a cost criterion that allows to automatically extract the most relevant features from the environment, leading to the "best" DBN representation. I discuss all building blocks required for a complete general learning algorithm.

Introduction

Agents. The agent-environment setup in which an *Agent* interacts with an *Environment* is a very general and prevalent framework for studying intelligent learning systems [RN03]. In cycles $t=1,2,3,...$, the environment provides a (regular) *observation* $o_t \in \mathcal{O}$ (e.g. a camera image) to the agent; then the agent chooses an *action* $a_t \in \mathcal{A}$ (e.g. a limb movement); finally the environment provides a real-valued *reward* $r_t \in \mathbb{R}$ to the agent. The reward may be very scarce, e.g. just +1 (-1) for winning (losing) a chess game, and 0 at all other times [Hut05, Sec.6.3]. Then the next cycle $t+1$ starts. The agent's objective is to maximize his reward.

Environments. For example, *sequence prediction* is concerned with environments that do not react to the agents actions (e.g. a weather-forecasting "action") [Hut03], *planning* deals with the case where the environmental function is known [RPPCd08], *classification* and *regression* is for conditionally independent observations [Bis06], *Markov Decision Processes* (MDPs) assume that o_t and r_t only depend on a_{t-1} and o_{t-1} [SB98], POMDPs deal with *Partially Observable MDPs* [KLC98], and *Dynamic Bayesian Networks* (DBNs) with structured MDPs [BDH99].

Feature MDPs [Hut09]. Concrete real-world problems can often be *modeled* as MDPs. For this purpose, a *designer* extracts relevant features from the history (e.g. position and velocity of all objects), i.e. the *history* $h_t = a_1 o_1 r_1 ... a_{t-1} o_{t-1} r_{t-1} o_t$ is summarized by a *feature* vector $s_t := \Phi(h_t)$. The feature vectors are regarded as *states* of an MDP and are assumed to be (approximately) Markov.

Artificial General Intelligence (AGI) [GP07] is concerned with designing *agents that perform well in a very large range of environments* [LH07], including all of the mentioned ones above and more. In this general situation, it is not a priori clear what the useful features are. Indeed, any observation in the (far) past may be relevant in the future. A solution suggested in [Hut09] is to learn Φ itself.

If Φ keeps too much of the history (e.g. $\Phi(h_t) = h_t$), the resulting MDP is too large (infinite) and cannot be learned. If Φ keeps too little, the resulting state sequence is not Markov. The *Cost* criterion I develop formalizes this trade-off and is minimized for the "best" Φ. At any time n, the best Φ is the one that minimizes the Markov code length of $s_1 ... s_n$ and $r_1 ... r_n$. This reminds but is actually quite different from MDL, which minimizes model+data code length [Grü07].

Dynamic Bayesian networks. The use of "unstructured" MDPs [Hut09], even our Φ-optimal ones, is clearly limited to relatively simple tasks. Real-world problems are structured and can often be represented by dynamic Bayesian networks (DBNs) with a reasonable number of nodes [DK89]. Bayesian networks in general and DBNs in particular are powerful tools for modeling and solving complex real-world problems. Advances in theory and increase in computation power constantly broaden their range of applicability [BDH99; SDL07].

Main contribution. The primary contribution of this work is to extend the Φ *selection principle* developed in [Hut09] for MDPs to the conceptually much more demanding DBN case. The major extra complications are approximating, learning and coding the rewards, the dependence of the Cost criterion on the DBN structure, learning the DBN structure, and how to store and find the optimal value function and policy.

Although this article is self-contained, it is recommended to read [Hut09] first.

Feature Dynamic Bayesian Networks (ΦDBN)

In this section I recapitulate the definition of ΦMDP from [Hut09], and adapt it to DBNs. While formally a DBN is just a special case of an MDP, exploiting the additional structure efficiently is a challenge. For generic MDPs, typical algorithms should be polynomial and can at best be linear in the number of states $|\mathcal{S}|$. For DBNs we want algorithms that are polynomial in the number of features m. Such DBNs

have exponentially many states ($2^{O(m)}$), hence the standard MDP algorithms are exponential, not polynomial, in m. Deriving poly-time (and poly-space!) algorithms for DBNs by exploiting the additional DBN structure is the challenge. The gain is that we can handle exponentially large structured MDPs efficiently.

Notation. Throughout this article, log denotes the binary logarithm, and $\delta_{x,y}=\delta_{xy}=1$ if $x=y$ and 0 else is the Kronecker symbol. I generally omit separating commas if no confusion arises, in particular in indices. For any z of suitable type (string,vector,set), I define string $\boldsymbol{z}=z_{1:l}=z_1...z_l$, sum $z_+=\sum_j z_j$, union $z_*=\bigcup_j z_j$, and vector $\boldsymbol{z}_\bullet=(z_1,...,z_l)$, where j ranges over the full range $\{1,...,l\}$ and $l=|z|$ is the length or dimension or size of z. $\hat{z}$ denotes an estimate of z. The characteristic function $\mathbb{1}_B=1$ if B=true and 0 else. $\mathrm{P}(\cdot)$ denotes a probability over states and rewards or parts thereof. I do not distinguish between random variables Z and realizations z, and abbreviation $\mathrm{P}(z):=\mathrm{P}[Z=z]$ never leads to confusion. More specifically, $m\in\mathbb{N}$ denotes the number of features, $i\in\{1,...,m\}$ any feature, $n\in\mathbb{N}$ the current time, and $t\in\{1,...,n\}$ any time. Further, due to space constraints at several places I gloss over initial conditions or special cases where inessential. Also 0∗undefined=0∗infinity:=0.

ΦMDP definition. A ΦMDP consists of a 7 tupel ($\mathcal{O}$,$\mathcal{A}$,$\mathcal{R}$,Agent,Env,Φ,$\mathcal{S}$) = (observation space, action space, reward space, agent, environment, feature map, state space). Without much loss of generality, I assume that $\mathcal{A}$ and $\mathcal{O}$ are finite and $\mathcal{R}\subseteq\mathbb{R}$. Implicitly I assume $\mathcal{A}$ to be small, while $\mathcal{O}$ may be huge.

Agent and Env are a pair or triple of interlocking functions of the history $\mathcal{H}:=(\mathcal{O}\times\mathcal{A}\times\mathcal{R})^*\times\mathcal{O}$:

$$\begin{aligned}\text{Env}: \mathcal{H}\times\mathcal{A}\times\mathcal{R}\rightsquigarrow\mathcal{O}, &\quad o_n=\text{Env}(h_{n-1}a_{n-1}r_{n-1}),\\ \text{Agent}: \mathcal{H}\rightsquigarrow\mathcal{A}, &\quad a_n=\text{Agent}(h_n),\\ \text{Env}: \mathcal{H}\times\mathcal{A}\rightsquigarrow\mathcal{R}, &\quad r_n=\text{Env}(h_na_n).\end{aligned}$$

where $\rightsquigarrow$ indicates that mappings $\to$ might be stochastic. The informal goal of AI is to design an Agent() that achieves high (expected) reward over the agent's lifetime in a large range of Env()ironments.

The feature map Φ maps histories to states

$$\Phi:\mathcal{H}\to\mathcal{S},\quad s_t=\Phi(h_t),\quad h_t=oar_{1:t-1}o_t\in\mathcal{H}$$

The idea is that Φ shall extract the "relevant" aspects of the history in the sense that "compressed" history $sar_{1:n}\equiv s_1a_1r_1...s_na_nr_n$ can well be described as a sample from some MDP ($\mathcal{S}$,$\mathcal{A}$,T,R) = (state space, action space, transition probability, reward function).

(Φ) Dynamic Bayesian Networks are structured (Φ)MDPs. The state space is $\mathcal{S}=\{0,1\}^m$, and each state $s\equiv\boldsymbol{x}\equiv(x^1,...,x^m)\in\mathcal{S}$ is interpreted as a feature vector $\boldsymbol{x}=\boldsymbol{\Phi}(h)$, where $x^i=\Phi^i(h)$ is the value of the ith binary feature. In the following I will also refer to x^i as feature i, although strictly speaking it is its value. Since non-binary features can be realized as a list of binary features, I restrict myself to the latter.

Given $\boldsymbol{x}_{t-1}=\boldsymbol{x}$, I assume that the features $(x_t^1,...,x_t^m)=\boldsymbol{x}'$ at time t are independent, and that each x'^i depends only on a subset of "parent" features $\boldsymbol{u}^i\subseteq\{x^1,...,x^m\}$, i.e. the transition matrix has the structure

$$T^a_{\boldsymbol{x}\boldsymbol{x}'}=\mathrm{P}(\boldsymbol{x}_t=\boldsymbol{x}'|\boldsymbol{x}_{t-1}=\boldsymbol{x},a_{t-1}=a)=\prod_{i=1}^m\mathrm{P}^a(x'^i|\boldsymbol{u}^i)\quad(1)$$

This defines our **ΦDBN model**. It is just a ΦMDP with special $\mathcal{S}$ and T. Explaining ΦDBN on an example is easier than staying general.

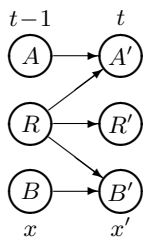
ΦDBN Example

Consider an instantiation of the simple vacuum world [RN03, Sec.3.6]. There are two rooms, A and B, and a vacuum Robot that can observe whether the room he is in is Clean or Dirty; Move to the other room, Suck, i.e. clean the room he is in; or do Nothing. After 3 days a room gets dirty again. Every clean room gives a reward 1, but a moving or sucking robot costs and hence reduces the reward by 1. Hence $\mathcal{O}=\{A,B\}\times\{C,D\}$, $\mathcal{A}=\{N,S,M\}$, $\mathcal{R}=\{-1,0,1,2\}$, and the dynamics Env() (possible histories) is clear from the above description.

Dynamics as a DBN. We can model the dynamics by a DBN as follows: The state is modeled by 3 features. Feature $R\in\{A,B\}$ stores in which room the robot is, and feature $A/B\in\{0,1,2,3\}$ remembers (capped at 3) how long ago the robot has cleaned room A/B last time, hence $\mathcal{S}=\{0,1,2,3\}\times\{A,B\}\times\{0,1,2,3\}$. The state/feature transition is as follows:

if ($x^R=A$ and $a=S$) then $x'^A=0$ else $x'^A=\min\{x^A+1,3\}$;
if ($x^R=B$ and $a=S$) then $x'^B=0$ else $x'^B=\min\{x^B+1,3\}$;
if $a=M$ (if $x^R=B$ then $x'^R=A$ else $x'^R=B$) else $x'^R=x^R$;

A DBN can be viewed as a two-layer Bayesian network [BDH99]. The dependency structure of our example is depicted in the right diagram.



Each feature consists of a (left,right)-pair of nodes, and a node $i\in\{1,2,3=m\}\widehat{=}\{A,R,B\}$ on the right is connected to all and only the parent features $\boldsymbol{u}^i$ on the left. The reward is

$$r=\mathbb{1}_{x^A<3}+\mathbb{1}_{x^B<3}-\mathbb{1}_{a\neq N}$$

The features map $\boldsymbol{\Phi}=(\Phi^A,\Phi^R,\Phi^B)$ can also be written down explicitly. It depends on the actions and observations of the last 3 time steps.

Discussion. Note that all nodes x'^i can implicitly also depend on the chosen action a. The optimal policies are repetitions of action sequence S,N,M or S,M,N. One might think that binary features $x^{A/B}\in\{C,D\}$ are sufficient, but this would result in a POMDP (Partially Observable MDP), since the cleanness of room A is not observed while the robot is in room B. That is, $\boldsymbol{x}'$ would not be a (probabilistic) function of $\boldsymbol{x}$ and a alone. The quaternary feature $x^A\in\{0,1,2,3\}$ can easily be converted into two binary features, and similarly x^B. The purely deterministic example can easily be made stochastic. For instance, Sucking and Moving may fail with a certain probability. Possible, but more complicated is to model a probabilistic transition from Clean to Dirty. In the randomized versions the agent needs to use its observations.

ΦDBN Coding and Evaluation

I now construct a code for $s_{1:n}$ given $a_{1:n}$, and for $r_{1:n}$ given $s_{1:n}$ and $a_{1:n}$, which is optimal (minimal) if $s_{1:n}r_{1:n}$ given $a_{1:n}$ is sampled from some MDP. It constitutes our cost function for $\mathbf{\Phi}$ and is used to define the $\mathbf{\Phi}$ selection principle for DBNs. Compared to the MDP case, reward coding is more complex, and there is an extra dependence on the graphical structure of the DBN.

Recall [Hut09] that a sequence $z_{1:n}$ with counts $\boldsymbol{n} = (n_1,...,n_m)$ can within an additive constant be coded in

$$\text{CL}(\boldsymbol{n}) := n\,H(\boldsymbol{n}/n) + \tfrac{m'-1}{2}\log n \;\text{ if }\; n>0 \;\text{ and }\; 0 \;\text{ else} \tag{2}$$

bits, where $n = n_+ = n_1+...+n_m$ and $m' = |\{i : n_i > 0\}| \leq m$ is the number of non-empty categories, and $H(\boldsymbol{p}) := -\sum_{i=1}^m p_i \log p_i$ is the entropy of probability distribution $\boldsymbol{p}$. The code is optimal (within $+O(1)$) for all i.i.d. sources.

State/Feature Coding. Similarly to the ΦMDP case, we need to code the temporal "observed" state=feature sequence $\boldsymbol{x}_{1:n}$. I do this by a frequency estimate of the state/feature transition probability. (Within an additive constant, MDL, MML, combinatorial, incremental, and Bayesian coding all lead to the same result). In the following I will drop the prime in $(\boldsymbol{u}^i,a,x'^i)$ tuples and related situations if/since it does not lead to confusion. Let $\mathcal{T}^{ia}_{\boldsymbol{u}^i x^i} = \{t \leq n : \boldsymbol{u}_{t-1} = \boldsymbol{u}^i, a_{t-1} = a, x^i_t = x^i\}$ be the set of times $t-1$ at which features that influence x^i have values $\boldsymbol{u}^i$, and action is a, and which leads to feature i having value x^i. Let $n^{ia}_{\boldsymbol{u}^i x^i} = |\mathcal{T}^{ia}_{\boldsymbol{u}^i x^i}|$ their number ($n^{i+}_{++} = n\;\forall i$). I estimate each feature probability separately by $\hat{\text{P}}^a(x^i|\boldsymbol{u}^i) = n^{ia}_{\boldsymbol{u}^i x^i}/n^{ia}_{\boldsymbol{u}^i +}$. Using (1), this yields

$$\begin{aligned}\hat{\text{P}}(\boldsymbol{x}_{1:n}|a_{1:n}) &= \prod_{t=1}^n \hat{T}^{a_{t-1}}_{\boldsymbol{x}_{t-1}\boldsymbol{x}_t} = \prod_{t=1}^n\prod_{i=1}^m \hat{\text{P}}^{a_{t-1}}(x^i_t|\boldsymbol{u}^i_{t-1}) \\ &= ... = \exp\Big[\sum_{i,\boldsymbol{u}^i,a} n^{ia}_{\boldsymbol{u}^i +} H\Big(\frac{\boldsymbol{n}^{ia}_{\boldsymbol{u}^i \bullet}}{n^{ia}_{\boldsymbol{u}^i +}}\Big)\Big]\end{aligned}$$

The length of the Shannon-Fano code of $\boldsymbol{x}_{1:n}$ is just the logarithm of this expression. We also need to code each non-zero count $n^{ia}_{\boldsymbol{u}^i x^i}$ to accuracy $O(1/\sqrt{n^{ia}_{\boldsymbol{u}^i +}})$, which each needs $\frac{1}{2}\log(n^{ia}_{\boldsymbol{u}^i +})$ bits. Together this gives a complete code of length

$$\text{CL}(\boldsymbol{x}_{1:n}|a_{1:n}) = \sum_{i,\boldsymbol{u}^i,a} \text{CL}(\boldsymbol{n}^{ia}_{\boldsymbol{u}^i \bullet}) \tag{3}$$

The rewards are more complicated.

Reward structure. Let $R^a_{\boldsymbol{x}\boldsymbol{x}'}$ be (a model of) the observed reward when action a in state $\boldsymbol{x}$ results in state $\boldsymbol{x}'$. It is natural to assume that the structure of the rewards $R^a_{\boldsymbol{x}\boldsymbol{x}'}$ is related to the transition structure $T^a_{\boldsymbol{x}\boldsymbol{x}'}$. Indeed, this is not restrictive, since one can always consider a DBN with the union of transition and reward dependencies. Usually it is assumed that the "global" reward is a *sum* of "local" rewards $R^{ia}_{\boldsymbol{u}^i x'^i}$, one for each feature i [KP99]. For simplicity of exposition I assume that the local reward R^i only depends on the feature value x'^i and not on $\boldsymbol{u}^i$ and a. Even this is not restrictive and actually may be advantageous as discussed in [Hut09] for MDPs. So I assume

$$R^a_{\boldsymbol{x}\boldsymbol{x}'} = \sum_{i=1}^m R^i_{x'^i} =: R(\boldsymbol{x}')$$

For instance, in the example in the previous section, two local rewards ($R^A_{x'^A} = \mathbb{1}_{x'^A<3}$ and $R^B_{x'^B} = \mathbb{1}_{x'^B<3}$) depend on $\boldsymbol{x}'$ only, but the third reward depends on the action ($R^R = -\mathbb{1}_{a\neq N}$).

Often it is assumed that the local rewards are directly observed or known [KP99], but we neither want nor can do this here: Having to specify many local rewards is an extra burden for the environment (e.g. the teacher), which preferably should be avoided. In our case, it is not even possible to pre-specify a local reward for each feature, since the features Φ^i themselves are learned by the agent and are not statically available. They are agent-internal and not part of the ΦDBN interface. In case multiple rewards *are* available, they can be modeled as part of the regular observations o, and r only holds the overall reward. The agent must and can learn to interpret and exploit the local rewards in o by himself.

Learning the reward function. In analogy to the MDP case for R and the DBN case for T above it is tempting to estimate $R^i_{x^i}$ by $\sum_{r'} r' n^{ir'}_{+x^i}/n^{i+}_{+x^i}$ but this makes no sense. For instance if $r_t = 1\;\forall t$, then $\hat{R}^i_{x^i} \equiv 1$, and $\hat{R}^a_{\boldsymbol{x}\boldsymbol{x}'} \equiv m$ is a gross mis-estimation of $r_t \equiv 1$. The localization of the global reward is somewhat more complicated. The goal is to choose $R^1_{x^1},...,R^m_{x^m}$ such that $r_t = R(\boldsymbol{x}_t)\;\forall t$.

Without loss we can set $R^i_0 \equiv 0$, since we can subtract a constant from each local reward and absorb them into an overall constant w_0. This allows us to write

$$R(\boldsymbol{x}) = w_0x^0 + w_1x^1 + ... + w_mx^m = \boldsymbol{w}^\top\boldsymbol{x}$$

where $w_i := R^i_1$ and $x^0 :\equiv 1$.

In practice, the ΦDBN model will not be perfect, and an approximate solution, e.g. a least squares fit, is the best we can achieve. The square loss can be written as

$$\text{Loss}(\boldsymbol{w}) := \sum_{t=1}^n (R(\boldsymbol{x}_t) - r_t)^2 = \boldsymbol{w}^\top A\boldsymbol{w} - 2\boldsymbol{b}^\top\boldsymbol{w} + c \tag{4}$$

$$A_{ij} := \sum_{t=1}^n x^i_t x^j_t, \quad b_i := \sum_{t=1}^n r_t x^i_t, \quad c := \sum_{t=1}^n r_t^2$$

Note that A_{ij} counts the number of times feature i and j are "on" (=1) simultaneously, and b_i sums all rewards for which feature i is on. The loss is minimized for

$$\hat{\boldsymbol{w}} := \arg\min_{\boldsymbol{w}} \text{Loss}(\boldsymbol{w}) = A^{-1}\boldsymbol{b}, \qquad \hat{R}(\boldsymbol{x}) = \hat{\boldsymbol{w}}^\top\boldsymbol{x}$$

which involves an inversion of the $(m+1)\times(m+1)$ matrix A. For singular A we take the pseudo-inverse.

Reward coding. The quadratic loss function suggests a Gaussian model for the rewards:

$$\text{P}(r_{1:n}|\hat{\boldsymbol{w}},\sigma) := \exp(-\text{Loss}(\hat{\boldsymbol{w}})/2\sigma^2)/(2\pi\sigma^2)^{n/2}$$

Maximizing this w.r.t. the variance σ^2 yields the maximum likelihood estimate

$$-\log\text{P}(r_{1:n}|\hat{\boldsymbol{w}},\hat{\sigma}) = \tfrac{n}{2}\log(\text{Loss}(\hat{\boldsymbol{w}})) - \tfrac{n}{2}\log\tfrac{n\text{e}}{2\pi}$$

where $\hat{\sigma}^2 = \text{Loss}(\hat{\boldsymbol{w}})/n$. Given $\hat{\boldsymbol{w}}$ and $\hat{\sigma}$ this can be regarded as the (Shannon-Fano) code length of $r_{1:n}$ (there are actually a few subtleties here which I gloss over). Each weight $\hat{w}_k$ and $\hat{\sigma}$ need also be coded to accuracy $O(1/\sqrt{n})$, which needs $(m+2)\frac{1}{2}\log n$ bits total. Together this gives a complete code of length

$$\begin{aligned}\text{CL}(r_{1:n}|\boldsymbol{x}_{1:n}a_{1:n}) \;&=\; \\ &=\; \tfrac{n}{2}\log(\text{Loss}(\hat{\boldsymbol{w}})) + \tfrac{m+2}{2}\log n - \tfrac{n}{2}\log\tfrac{ne}{2\pi}\end{aligned} \tag{5}$$

ΦDBN evaluation and selection is similar to the MDP case. Let G denote the graphical structure of the DBN, i.e. the set of parents $\text{Pa}^i \subseteq \{1,...,m\}$ of each feature i. (Remember $\boldsymbol{u}^i$ are the parent values). Similarly to the MDP case, the cost of $(\boldsymbol{\Phi},G)$ on h_n is defined as

$$\text{Cost}(\boldsymbol{\Phi},G|h_n) \;:=\; \text{CL}(\boldsymbol{x}_{1:n}|a_{1:n}) + \text{CL}(r_{1:n}|\boldsymbol{x}_{1:n},a_{1:n}), \tag{6}$$

and the best $(\boldsymbol{\Phi},G)$ minimizes this cost.

$$(\boldsymbol{\Phi}^{best},G^{best}) \;:=\; \arg\min_{\boldsymbol{\Phi},G}\{\text{Cost}(\boldsymbol{\Phi},G|h_n)\}$$

A general discussion why this is a good criterion can be found in [Hut09]. In the following section I mainly highlight the difference to the MDP case, in particular the additional dependence on and optimization over G.

DBN Structure Learning & Updating

This section briefly discusses minimization of (6) w.r.t. G given $\boldsymbol{\Phi}$ and even briefer minimization w.r.t. $\boldsymbol{\Phi}$. For the moment regard $\boldsymbol{\Phi}$ as given and fixed.

Cost and DBN structure. For general structured local rewards $R^{ia}_{\boldsymbol{u}^i x'^i}$, (3) and (5) both depend on G, and (6) represents a novel DBN structure learning criterion that includes the rewards.

For our simple reward model $R^i_{x^i}$, (5) is independent of G, hence only (3) needs to be considered. This is a standard MDL criterion, but I have not seen it used in DBNs before. Further, the features i are independent in the sense that we can search for the optimal parent sets $\text{Pa}^i \subseteq \{1,...,m\}$ for each feature i separately.

Complexity of structure search. Even in this case, finding the optimal DBN structure is generally hard. In principle we could rely on off-the-shelf heuristic search methods for finding good G, but it is probably better to use or develop some special purpose optimizer. One may even restrict the space of considered graphs G to those for which (6) can be minimized w.r.t. G efficiently, as long as this restriction can be compensated by "smarter" $\boldsymbol{\Phi}$.

A brute force exhaustive search algorithm for Pa^i is to consider all 2^m subsets of $\{1,...,m\}$ and select the one that minimizes $\sum_{\boldsymbol{u}^i,a}\text{CL}(\boldsymbol{n}^{ia}_{\boldsymbol{u}^i\bullet})$. A reasonable and often employed assumption is to limit the number of parents to some small value p, which reduces the search space size to $O(m^p)$.

Indeed, since the Cost is exponential in the maximal number of parents of a feature, but only linear in n, a Cost minimizing Φ can usually not have more than a logarithmic number of parents, which leads to a search space that is pseudo-polynomial in m.

Heuristic structure search. We could also replace the well-founded criterion (3) by some heuristic. One such heuristic has been developed in [SDL07]. The mutual information is another popular criterion for determining the dependency of two random variables, so we could add j as a parent of feature i if the mutual information of x^j and x'^i is above a certain threshold. Overall this takes time $O(m^2)$ to determine G. An MDL inspired threshold for binary random variables is $\frac{1}{2n}\log n$. Since the mutual information treats parents independently, $\hat{T}$ has to be estimated accordingly, essentially as in naive Bayes classification [Lew98] with feature selection, where x'^i represents the class label and $\boldsymbol{u}^i$ are the features selected $\boldsymbol{x}$. The improved Tree-Augmented naive Bayes (TAN) classifier [FGG97] could be used to model synchronous feature dependencies (i.e. within a time slice). The Chow-Liu [CL68] minimum spanning tree algorithm allows determining G in time $O(m^3)$. A tree becomes a forest if we employ a lower threshold for the mutual information.

$\boldsymbol{\Phi}$ search is even harder than structure search, and remains an art. Nevertheless the reduction of the complex (ill-defined) reinforcement learning problem to an internal feature search problem with well-defined objective is a clear conceptual advance.

In principle (but not in practice) we could consider the set of *all* (computable) functions $\{\Phi : \mathcal{H} \to \{0,1\}\}$. We then compute $\text{Cost}(\boldsymbol{\Phi}|h)$ for every finite subset $\boldsymbol{\Phi} = \{\Phi^{i_1},...,\Phi^{i_m}\}$ and take the minimum (note that the order is irrelevant).

Most practical search algorithms require the specification of some neighborhood function, here for $\boldsymbol{\Phi}$. For instance, stochastic search algorithms suggest and accept a neighbor of $\boldsymbol{\Phi}$ with a probability that depends on the Cost reduction. See [Hut09] for more details. Here I will only present some very simplistic ideas for features and neighborhoods.

Assume binary observations $\mathcal{O} = \{0,1\}$ and consider the last m observations as features, i.e. $\Phi^i(h_n) = o_{n-i+1}$ and $\boldsymbol{\Phi}(h_n) = (\Phi^1(h_n),...,\Phi^m(h_n)) = o_{n-m+1:n}$. So the states are the same as for Φ_mMDP in [Hut09], but now $\mathcal{S} = \{0,1\}^m$ is structured as m binary features. In the example here, $m = 5$ lead to a perfect ΦDBN. We can add a new feature o_{n-m} ($m \rightsquigarrow m+1$) or remove the last feature ($m \rightsquigarrow m-1$), which defines a natural neighborhood structure.

Note that the context trees of [McC96; Hut09] are more flexible. To achieve this flexibility here we either have to use smarter features within our framework (simply interpret $s = \Phi_{\mathcal{S}}(h)$ as a feature vector of length $m = \lceil\log|\mathcal{S}|\rceil$) or use smarter (non-tabular) estimates of $\text{P}^a(x^i|\boldsymbol{u}^i)$ extending our framework (to tree dependencies).

For general purpose intelligent agents we clearly need more powerful features. Logical expressions or (non)accepting Turing machines or recursive sets can map histories or parts thereof into true/false or accept/reject or in/out, respectively, hence naturally represent binary features. Randomly generating such expressions or programs with an appropriate bias towards simple ones is a universal feature generator that eventually finds the optimal feature map. The idea is known as Universal Search [Gag07].

Value & Policy Learning in ΦDBN

Given an estimate $\hat{\mathbf{\Phi}}$ of $\mathbf{\Phi}^{best}$, the next step is to determine a good action for our agent. I mainly concentrate on the difficulties one faces in adapting MDP algorithms and discuss state of the art DBN algorithms. Value and policy learning in known finite state MDPs is easy provided one is satisfied with a polynomial time algorithm. Since a DBN is just a special (structured) MDP, its (Q) Value function respects the same Bellman equations [Hut09, Eq.(6)], and the optimal policy is still given by $a_{n+1} := \text{argmax}_a Q^{*a}_{\boldsymbol{x}_{n+1}}$. Nevertheless, their solution is now a nightmare, since the state space is exponential in the number of features. We need algorithms that are polynomial in the number of features, i.e. logarithmic in the number of states.

Value function approximation. The first problem is that the optimal value and policy do not respect the structure of the DBN. They are usually complex functions of the (exponentially many) states, which cannot even be stored, not to mention computed [KP99]. It has been suggested that the value can often be approximated well as a sum of local values similarly to the rewards. Such a value function can at least be stored.

Model-based learning. The default quality measure for the approximate value is the ρ-weighted squared difference, where ρ is the stationary distribution.

Even for a fixed policy, value iteration does *not* converge to the best approximation, but usually converges to a fixed point close to it [BT96]. Value iteration requires ρ explicitly. Since ρ is also too large to store, one has to approximate ρ as well. Another problem, as pointed out in [KP00], is that policy iteration may not converge, since different policies have different (misleading) stationary distributions. Koller and Parr [KP00] devised algorithms for general factored ρ, and Guestrin et al. [GKPV03] for max-norm, alleviating this problem. Finally, general policies cannot be stored exactly, and another restriction or approximation is necessary.

Model-free learning. Given the difficulties above, I suggest to (re)consider a very simple class of algorithms, without suggesting that it is better. The above model-based algorithms exploit $\hat{T}$ and $\hat{R}$ directly. An alternative is to sample from $\hat{T}$ and use model-free "Temporal Difference (TD)" learning algorithms based only on this internal virtual sample [SB98]. We could use TD(λ) or Q-value variants with linear value function approximation.

Beside their simplicity, another advantage is that neither the stationary distribution nor the policy needs to be stored or approximated. Once approximation $\hat{Q}^*$ has been obtained, it is trivial to determine the optimal (w.r.t. $\hat{Q}^*$) action via $a_{n+1} = \text{argmax}_a Q^{*a}_{\boldsymbol{x}_{n+1}}$ for any state of interest (namely $\boldsymbol{x}_{n+1}$) exactly.

Exploration. Optimal actions based on approximate rather than exact values can lead to very poor behavior due to lack of exploration. There are polynomially optimal algorithms (Rmax,E3,OIM) for the exploration-exploitation dilemma.

For model-based learning, extending E3 to DBNs is straightforward, but E3 needs an oracle for planning in a given DBN [KK99]. Recently, Strehl et al. [SDL07] accomplished the same for Rmax. They even learn the DBN structure, albeit in a very simplistic way. Algorithm OIM [SL08], which I described in [Hut09] for MDPs, can also likely be generalized to DBNs, and I can imagine a model-free version.

Incremental Updates

As discussed two sections ago, most search algorithms are local in the sense that they produce a chain of "slightly" modified candidate solutions, here $\mathbf{\Phi}$'s. This suggests a potential speedup by computing quantities of interest incrementally.

Cost. Computing $\text{CL}(\boldsymbol{x}|\boldsymbol{a})$ in (3) takes at most time $O(m2^k|\mathcal{A}|)$, where k is the maximal number of parents of a feature. If we remove feature i, we can simply remove/subtract the contributions from i in the sum. If we add a new feature $m+1$, we only need to search for the best parent set $\boldsymbol{u}^{m+1}$ for this new feature, and add the corresponding code length. In practice, many transitions don't occur, i.e. $n^{ia}_{\boldsymbol{u}^i x^i}=0$, so $\text{CL}(\boldsymbol{x}|\boldsymbol{a})$ can actually be computed much faster in time $O(|\{n^{ia}_{\boldsymbol{u}^i x^i}>0\}|)$, and incrementally even faster.

Rewards. When adding a new feature, the current local reward estimates may not change much. If we reassign a fraction $\alpha \leq 1$ of reward to the new feature x^{m+1}, we get the following ansatz[1].

$$\hat{R}(x^1,...,x^{m+1}) = (1-\alpha)\hat{R}(\boldsymbol{x}) + w_{m+1}x^{m+1} =: \boldsymbol{v}^\top \boldsymbol{\psi}(\boldsymbol{x})$$
$$\boldsymbol{v} \;:=\; (1-\alpha, w_{m+1})^\top, \quad \boldsymbol{\psi} \;:=\; (\hat{R}(\boldsymbol{x}), x^{m+1})^\top$$

Minimizing $\sum_{t=1}^n (\hat{R}(x_t^1...x_t^{m+1}) - r_t)^2$ w.r.t. $\boldsymbol{v}$ analogous to (4) just requires a trivial 2×2 matrix inversion. The minimum $\tilde{\boldsymbol{v}}$ results in an initial new estimate $\tilde{\boldsymbol{w}} = ((1-\tilde{\alpha})\hat{w}_0,...,(1-\tilde{\alpha})\hat{w}_m, \tilde{w}_{m+1})^\top$, which can be improved by some first order gradient decent algorithm in time $O(m)$, compared to the exact $O(m^3)$ algorithm. When removing a feature, we simply redistribute its local reward to the other features, e.g. uniformly, followed by improvement steps that cost $O(m)$ time.

Value. All iteration algorithms described in the previous section for computing (Q) Values need an initial value for V or Q. We can take the estimate $\hat{V}$ from a previous $\mathbf{\Phi}$ as an initial value for the new $\mathbf{\Phi}$. Similarly as for the rewards, we can redistribute a fraction of the values by solving relatively small systems of equations. The result is then used as an initial value for the iteration algorithms in the previous section. A further speedup can be obtained by using prioritized iteration algorithms that concentrate their time on badly estimated parameters, which are in our case the new values [SB98].

Similarly, results from time t can be (re)used as initial estimates for the next cycle $t+1$, followed by a fast improvement step.

Outlook

ΦDBN leaves much more questions open and room for modifications and improvements than ΦMDP. Here are a few.

[1] An *Ansatz* is an initial mathematical or physical model with some free parameters to be determined subsequently. [http://en.wikipedia.org/wiki/Ansatz]

- The cost function can be improved by integrating out the states analogous to the ΦMDP case [Hut09]: The likelihood $\mathrm{P}(r_{1:n}|a_{1:n},\hat{U})$ is unchanged, except that $\hat{U}\equiv\hat{T}\hat{R}$ is now estimated locally, and the complexity penalty becomes $\frac{1}{2}(M+m+2)\log n$, where M is (essentially) the number of non-zero counts $n^{ia}_{\boldsymbol{u}^i x^i}$, but an efficient algorithm has yet to be found.
- It may be necessary to impose and exploit structure on the conditional probability tables $P^a(x^i|\boldsymbol{u}^i)$ themselves [BDH99].
- Real-valued observations and beliefs suggest to extend the binary feature model to $[0,1]$ interval valued features rather than coding them binary. Since any continuous semantics that preserves the role of 0 and 1 is acceptable, there should be an efficient way to generalize Cost and *V*alue estimation procedures.
- I assumed that the reward/value is linear in local rewards/values. Is this sufficient for all practical purposes? I also assumed a least squares and Gaussian model for the local rewards. There are efficient algorithms for much more flexible models. The least we could do is to code w.r.t. the proper covariance A.
- I also barely discussed synchronous (within time-slice) dependencies.
- I guess ΦDBN will often be able to work around too restrictive DBN models, by finding features Φ that are more compatible with the DBN and reward structure.
- Extra edges in the DBN can improve the linear value function approximation. To give ΦDBN incentives to do so, the *V*alue would have to be included in the Cost criterion.
- Implicitly I assumed that the action space $\mathcal{A}$ is small. It is possible to extend ΦDBN to large structured action spaces.
- Apart from the Φ-search, all parts of ΦDBN seem to be poly-time approximable, which is satisfactory in theory. In practice, this needs to be improved to essentially linear time in n and m.
- Developing smart Φ generation and smart stochastic search algorithms for Φ are the major open challenges.
- A more Bayesian Cost criterion would be desirable: a likelihood of h given Φ and a prior over Φ leading to a posterior of Φ given h, or so. Monte Carlo (search) algorithms like Metropolis-Hastings could sample from such a posterior. Currently probabilities ($\widehat{=}2^{-\mathrm{CL}}$) are assigned only to rewards and states, but not to observations and feature maps.

Summary. In this work I introduced a powerful framework (ΦDBN) for general-purpose intelligent learning agents, and presented algorithms for all required building blocks. The introduced cost criterion reduced the informal reinforcement learning problem to an internal well-defined search for "relevant" features.

References

[BDH99] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[BT96] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[CL68] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467, 1968.

[DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[FGG97] N. Friedman, D. Geiger, and M. Goldszmid. Bayesian network classifiers. *Machine Learning*, 29(2):131–163, 1997.

[Gag07] M. Gaglio. Universal search. *Scholarpedia*, 2(11):2575, 2007.

[GKPV03] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research (JAIR)*, 19:399–468, 2003.

[GP07] B. Goertzel and C. Pennachin, editors. *Artificial General Intelligence*. Springer, 2007.

[Grü07] P. D. Grünwald. *The Minimum Description Length Principle*. The MIT Press, Cambridge, 2007.

[Hut03] M. Hutter. Optimality of universal Bayesian prediction for general loss and alphabet. *Journal of Machine Learning Research*, 4:971–1000, 2003.

[Hut05] M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2005. 300 pages, http://www.hutter1.net/ai/uaibook.htm.

[Hut09] M. Hutter. Feature Markov decision processes. In *Artificial General Intelligence (AGI'09)*. Atlantis Press, 2009.

[KK99] M. Kearns and D. Koller. Efficient reinforcement learning in factored MDPs. In *Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 740–747, San Francisco, 1999. Morgan Kaufmann.

[KLC98] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[KP99] D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs,. In *Proc. 16st International Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 1332–1339, Edinburgh, 1999.

[KP00] D. Koller and R. Parr. Policy iteration for factored MDPs. In *Proc. 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 326–334, San Francisco, CA, 2000. Morgan Kaufmann.

[Lew98] D. D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In *Proc. 10th European Conference on Machine Learning (ECML'98)*, pages 4–15, Chemnitz, DE, 1998. Springer.

[LH07] S. Legg and M. Hutter. Universal intelligence: A definition of machine intelligence. *Minds & Machines*, 17(4):391–444, 2007.

[McC96] A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, 1996.

[RN03] S. J. Russell and P. Norvig. *Artificial Intelligence. A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003.

[RPPCd08] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 2008(32):663–704, 2008.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[SDL07] A. L. Strehl, C. Diuk, and M. L. Littman. Efficient structure learning in factored-state MDPs. In *Proc. 27th AAAI Conference on Artificial Intelligence*, pages 645–650, Vancouver, BC, 2007. AAAI Press.

[SL08] I. Szita and A. Lörincz. The many faces of optimism: a unifying approach. In *Proc. 12th International Conference (ICML 2008)*, volume 307, Helsinki, Finland, June 2008.