Sampling in Factored Dynamic Systems

Daphne Koller and Uri Lerner

1 Introduction

In many real-world domains, we are interested in monitoring the evolution of a complex situation over time. For example, we may be monitoring a patient's vital signs in an intensive care unit (Dagum and Galper 1995), analyzing a complex freeway traffic scene with the goal of controlling a moving vehicle (Huang, Koller, Malik, Ogasawara, Rao, Russell and Weber 1994), localizing a robot in a complex environment (Fox, Burgard and Thrun 1999, Murphy and Russell 2000), or tracking motion of non-rigid objects in a cluttered visual scene (Isard and Blake 1998). We model such systems as being in one of a possible set of states, where the state changes over time. We model the states as changing at discrete time intervals, so that $\mathbf{x}_t$ is the state of the system at time t. In most systems, we model the system states as having some internal structure: the system state is typically represented via some vector of *variables* $\mathbf{X} = (X_1, \ldots, X_n)$, where each X_i takes on values in some space $Dom[X_i]$. The possible states $\mathbf{x}$ are assignments of values to the variables $\mathbf{X}$. In a traffic surveillance application, the state might contain variables such as the vehicle position, its velocity, the weather, and more.

In most cases, the system dynamics are unpredictable. We model such systems as a stochastic dynamic system, where we encode a probability distribution over all possible trajectories. By making the assumptions that the system is Markovian and time-invariant, we can represent this distribution using two components: a *prior probability distribution* π_0, which represents the distribution over the initial state of the system; and a *transition model* $p(\mathbf{X}' \mid \mathbf{X})$ which represents the probability that the system will transition from state $\mathbf{X}$ to state $\mathbf{X}'$ in a single step.[1]

Dynamic Bayesian networks (DBNs) (Dean and Kanazawa 1989) are a representation language for stochastic dynamic systems that allows us to represent complex systems in a compact and natural way. DBNs exploit certain assumptions about locality of interaction that are very common in real-world

[1] We use boldface to distinguish between a set of random variables $\mathbf{X}$ and particular variable X. We use upper case letters X (or $\mathbf{X}$) to denote random variables, such as the (unknown) state of the process at a given time, and lower case letters x (or $\mathbf{x}$) to denote particular instantiations to these variables such as a particular state of the system.

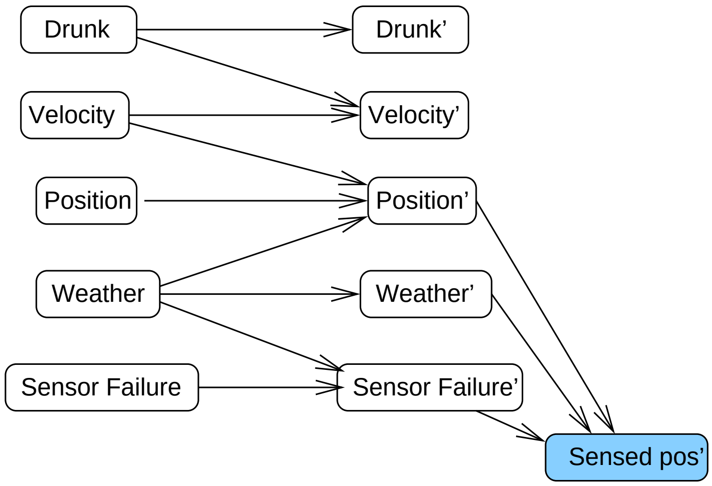


Figure 1: A highly simplified DBN for traffic surveillance.

systems. For example, in a freeway traffic surveillance domain, we might reasonably assume that a vehicle's position at time t depends only on its position at time $t-1$ and its velocity at time $t-1$. The vehicle's forward velocity at time t might depend on the road condition and on the clearance to the vehicle in front at time $t-1$. Like a Bayesian network (BN) (Pearl 1988), a DBN utilizes a graphical notation to represent the direct dependencies between the variables in the model. The transition model of the system — $p(\mathbf{X}' \mid \mathbf{X})$ — is represented in a factorized way using a network fragment such as the one in Figure 1, appropriately annotated with probabilities.

DBNs provide a convenient and compact representation that allows us to model even very large and complex systems, including very large discrete systems and systems that include both discrete and continuous variables. Hidden Markov models are a very simple special case of DBNs, as are linear Gaussian systems (Kalman 1960, Bar-Shalom and Fortmann 1988). DBNs have been used for a variety of applications, including freeway surveillance (Huang et al. 1994), complex factories (Jensen, Kjærulff, Olesen and Pedersen 1989), robotics (Nicholson and Brady 1994), medical monitoring (Dagum and Galper 1995), speech recognition (Zweig and Russell 1998), and more.

Of course, modeling complex systems is only the first step; we also need to use these models for inference. A common goal in dynamic systems is *tracking* (also called *filtering* or *monitoring*). At each time point, we get an observation $\mathbf{y}_t$, which is a partial observation of the state of the system. We wish to keep track of the state of the system, based on a partial observation $\mathbf{y}_t$. As our observation does not completely determine the true state, the best we can do is to keep track of $p(\mathbf{X}_t \mid \mathbf{y}_1, \ldots, \mathbf{y}_t)$. We call this distribution the *belief state at time t*.

In static Bayesian networks (ones that represent a probability distribution over a fixed finite set of variables), the model structure can be exploited to allow effective inference. Unfortunately, we can show that the same phenomenon does not apply to dynamic systems. Even in highly structured DBNs, the belief state is entirely unstructured (except in degenerate cases) (Boyen and Koller 1998). Hence, we can only do exact tracking by maintaining our belief state as a full joint distribution over the possible states. In discrete DBNs, the cost is exponential in the number of state variables. In continuous and hybrid DBNs, the situation is even worse. In most systems (Kalman filters being the only notable exception), the complexity of the belief state grows unboundedly over time, preventing any closed-form representation.

There have been several approximate inference techniques proposed for DBNs (Ghahramani and Jordan 1997, Boyen and Koller 1998), but they are designed primarily for discrete domains (or, at best, a very limited class of hybrid domains). Sequential Monte Carlo methods (Handschin and Mayne 1969, Gordon, Salmond and Smith 1993), originally proposed for DBNs under the name *survival of the fittest* (Kanazawa, Koller and Russell 1995) are currently the only approach that allows us to perform tracking in general purpose hybrid DBN models.

The remainder of the paper is structured as follows. In Section 2, we review the representation and semantics of DBN models. In Section 3, we show how particle filtering can be applied to general DBN models. In Section 4, we present experimental results for particle filtering in two large-scale DBN models, one of them discrete and the other hybrid. We conclude in Section 5 with some discussion and directions for future work.

2 Structured probabilistic models

2.1 Bayesian networks

Bayesian networks are a representation for probability distributions over complex domains. We consider probability spaces defined as the set of possible assignments to some set of random variables $X_1, \ldots, X_n$, each of which has a domain $Dom[X_i]$ of possible values. For example, we might have that the domain of the discrete variable *Weather* is {*clear, rain, snow*}. The domain of the variable *Velocity* can be $\mathbb{R}$, or it can be discretized into some appropriate partition. The goal is to represent a joint probability distribution over these variables.

Unfortunately, in discrete networks, an explicit description of the joint distribution requires a number of parameters that is exponential in n, the number of variables. Bayesian networks (Pearl 1988) derive their power from the ability to represent conditional independences among variables, which allows them to take advantage of the "locality" of causal influences. Intuitively, a variable is independent of its indirect causal influences given its

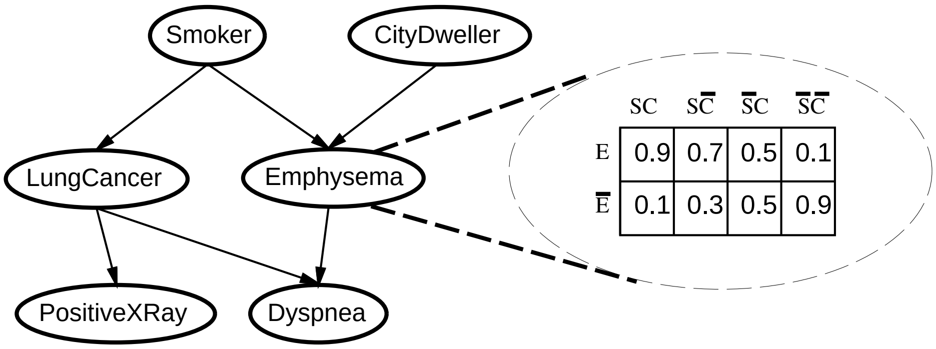


Figure 2: A simple Bayesian network, with a CPD for one of the nodes, specifying the conditional probability of each possible value of the variable *Emphysema*, given each possible combination of values of the parent nodes *Smoker* and *CoalMiner*.

direct causal influences. In Figure 2, for example, the outcome of the X-ray does not depend on whether the patient is a smoker given that we know that the patient has lung cancer. If each variable has at most k other variables that directly influence it, then the total number of required parameters is linear in n and exponential in k. This enables the compact representation of fairly large problems.

Formally, a Bayesian network is defined by a directed acyclic graph together with a local probabilistic model for each node. There is a node in the graph for each random variable $X_1, \ldots, X_n$. The edges in the graph denote direct dependency of a variable X_i on its parents $Parents(X_i)$. More formally, the graphical structure encodes a set of conditional independence assumptions: each node X_i is conditionally independent of its non-descendants given its parents.

The local probabilistic model for a node in the graph encodes the quantitative relationship between the node and its parents. It consists of a *conditional probability distribution* (CPD) $p(X_i \mid Parents(X_i))$, which specifies the distribution over the values of X_i given any possible assignment of values to its parents.

The qualitative independence assumptions implied by the network structure, combined with the CPDs associated with the nodes, are enough to specify a full joint distribution through the following equation, known as the *chain rule for Bayesian networks*:

$$p(X_1 \ldots X_n) = \prod_{i=1}^{n} p(X_i \mid Parents(X_i)). \tag{2.1}$$

2.2 Hybrid networks

While the CPD is often represented as a simple table, as in Figure 2, it can also be represented implicitly as a parameterized function from values of $Parents(X_i)$ to probability distributions over X_i. With implicitly represented CPDs, a network can include continuous as well as discrete variables.

As an example, we might represent a conditional distribution of a continuous node with a discrete parent as a conditional Gaussian. Formally, for a variable X with parent set $\mathbf{U}$, we can specify a CPD as follows: for every value $\mathbf{u} \in Dom[\mathbf{U}]$, the CPD has a parameter $\mu_{\mathbf{u}}$ and $\sigma^2_{\mathbf{u}}$; the conditional distribution is then:

$$p(X \mid \mathbf{u}) = N(\mu_{\mathbf{u}}; \sigma^2_{\mathbf{u}}).$$

For a continuous node X with continuous parents $Y_1, \ldots, Y_k$, we might choose to model X as a linear Gaussian of its parents. In this case, we would specify the CPD via parameters $a_0, \ldots, a_k$ and σ^2, and have

$$p(X \mid y_1, \ldots, y_k) = N(a_0 + a_1 y_1 + \cdots + a_k y_k; \sigma^2)$$

The linear Gaussian model is a very natural model; it is also very useful because there is an equivalence between BNs with linear Gaussian models and multivariate Gaussian distributions: every continuous BN all of whose CPDs are linear Gaussian defines a multivariate Gaussian distribution; conversely, every multivariate Gaussian distribution can be represented as a continuous BN with linear Gaussian CPDs (Shachter and Kenley 1989).

We can extend a linear Gaussian CPD to allow the continuous node to have discrete parents. The idea is the same as the one used above. If a node X has continuous parents $\mathbf{Y}$ and discrete parents $\mathbf{U}$, we could have a set of different linear Gaussians for X as a function of $\mathbf{Y}$ — one linear Gaussian (with its own parameters) for each value $\mathbf{u}$ of $\mathbf{U}$. This dependency model is called a *conditional linear Gaussian*. A BN whose discrete nodes have only discrete parents and whose continuous nodes have conditional linear Gaussian CPDs induces a joint distribution that defines a multivariate Gaussian distribution over the continuous variables for every instantiation to the discrete network variables.

We can also accomodate the case of a discrete child with a continuous parent. One reasonable model is based on the *softmax* density (Bishop 1995). Intuitively, the *softmax CPD* (Koller, Lerner and Angelov 1999) defines a set of R regions (for some parameter R of our choice). The regions are defined by a set of R linear functions over the continuous variables. A region is characterized as that part of the space where one particular linear function is bigger than all the others. Each region is also associated with some distribution over the values of the discrete child; this distribution is the one used for the variable within this region. The actual CPD is a continuous version of this region-based idea, allowing for smooth transitions between the distributions in neighboring regions of the space.

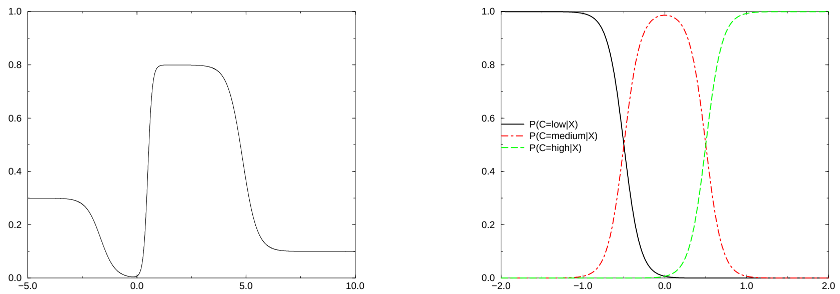
More precisely, let U be a discrete variable, with continuous parents $\mathbf{Y} = \{Y_1, \ldots, Y_k\}$. Assume that U has m possible values, $\{u_1, u_2, \ldots, u_m\}$. Each of the R regions is defined via two vectors of parameters $\alpha^r, \mathbf{p}^r$. The vector α^r is a vector of weights $\alpha_0^r, \alpha_1^r, \ldots, \alpha_k^r$ specifying the linear function associated with the region. The vector $\mathbf{p}^i = \{p_1^r, \ldots, p_m^r\}$ is the probability distribution over $u_1, \ldots, u_m$ associated with the region (i.e., $\sum_{j=1}^m p_j^r = 1$). The CPD is now defined as:

$$\begin{aligned} p(U = u_j \mid \mathbf{Y}) &= \sum_{r=1}^{R} w^r p_j^r \\ w^r &= \frac{\exp(\alpha_0^r + \sum_{i=1}^k \alpha_i^r Y_i)}{\sum_{q=1}^R \exp(\alpha_0^q + \sum_{i=1}^k \alpha_i^q Y_i)} \end{aligned}$$

In other words, the distribution is a weighted average of the region distributions, where the weight of each "region" depends exponentially on how big the value of its defining linear function is, relative to the rest. The choice of α^i determines both the regions and the slope of the transitions between them; the choice of $\mathbf{p}^i$ determines the distribution defining each region. Like the conditional linear Gaussian CPD, if U also has discrete parents, our softmax CPD will have a separate component for each instantiation of the discrete parents.

The power to choose the number of regions R to be as large as we wish is the key to the rich expressive power of the generalized softmax CPD. Figure 3 demonstrates this expressivity. In Figure 3(a), we present an example CPD for a binary variable with $R = 4$ regions. In Figure 3(b), we show how this CPD can be used to represent a simple classifier. Here, U is a sensor with three values: low, medium and high. The probability of each of these values depends on the value of the continuous parent X. Note that we can easily accomodate a variety of noise models for the sensor: we can make it less reliable in borderline situations by making the transitions between regions more moderate; we can make it inherently more noisy by having the probabilities of the different values in each of the regions be farther away from 0 and 1.

We have presented a small set of CPD models that are useful across a wide range of applications. Of course, there is an unlimited range of representations. For example, the model of (Fox et al. 1999) for robot localization uses a linear Gaussian CPD to represent the distribution of the sonar sensor value given the position of a known obstacle, and a very different parameterized CPD (an exponential distribution) to represent the distribution if there is no known obstacle. In principle, we can use parametric representation for a function of the appropriate type; we are limited only by how rich we choose to make our language for specifying CPDs. (The BUGS system (Gilks, Richardson and Spiegelhalter 1996) provides a very rich language for this purpose.)



(a) (b)

Figure 3: Expressive power of a softmax CPD.

2.3 Dynamic Bayesian networks

A *dynamic Bayesian network* uses the same ideas as a BN for modeling dynamic systems. As stated in the introduction, we typically assume that the process is *Markovian* so that the probability of being in state $\mathbf{x}'$ at time $t+1$ depends only on the state of the world at time t. We also typically assume that the process is time invariant, so that $p(\mathbf{X}_{t+1} \mid \mathbf{X}_t)$ does not depend on t. Thus, it can be specified using a single *transition model* which holds for all time points.

In a DBN, the state of the process at any given time is specified in terms of a set of state variables $X_1, \ldots, X_n$. We specify both the prior distribution π_0 and the transition model $p_\rightarrow(\mathbf{X}' \mid \mathbf{X})$ using a compact graphical representation.

Definition 2.1: A *2-time-slice Bayesian network (2-TBN)* for a process whose state variables are $\mathbf{X}$ is a Bayesian network *fragment* defined as follows. The graph structure is a directed acyclic graph whose nodes are $\mathbf{X} \cup \mathbf{X}'$, where nodes in $\mathbf{X}$ have no parents. The nodes in $\mathbf{X}'$ are annotated with CPDs, $p(X' \mid Parents(X'))$. The 2-TBN represents a *conditional distribution* $p_\rightarrow(\mathbf{X}' \mid \mathbf{X})$ via the *chain rule for 2-TBNs*:

$$p_\rightarrow(\mathbf{X}' \mid \mathbf{X}) = \prod_{i=1}^{n} p(X'_i \mid Parents(X'_i)).$$

∎

A 2-TBN for a simple vehicle surveillance problem was shown in Figure **??**. Note that a 2-TBN represents a conditional distribution; hence, it only specifies a probabilistic model for the variables $\mathbf{X}'$: for each variable X'_i, it specifies a set of parents $Parents(X'_i)$, which can be variables either in the previous time slice or the current one, and a CPD. The 2-TBN model allows us to encode stronger independence assumptions than the standard Markov property:

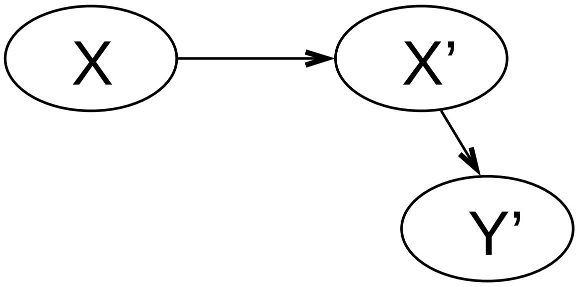


Figure 4: The representation of an HMM or Kalman filter as a DBN.

a variable X'_i depends only on its immediate parents, i.e., it is conditionally independent of all other variables in the past given values for its parents.

Definition 2.2: A *dynamic Bayesian network* is a pair $\langle \mathcal{B}_0, \mathcal{B}_\rightarrow \rangle$, where $\mathcal{B}_0$ is a Bayesian network over $\mathbf{X}_0$, representing the initial distribution π_0 over the initial state, and $\mathcal{B}_\rightarrow$ is a 2-TBN for the process representing the transition model $p_\rightarrow$. For any T, the distribution over $\mathbf{X}_0, \ldots, \mathbf{X}_T$ is defined as:

$$p(\mathbf{X}_0 = \mathbf{x}_0, \ldots, \mathbf{X}_T = \mathbf{x}_T) = \pi_0(\mathbf{x}_0) \cdot \prod_{t=1}^{T} p_\rightarrow(\mathbf{X}' = \mathbf{x}_t \mid \mathbf{X} = \mathbf{x}_{t-1}).$$

∎

In most dynamic systems, we have an *observation model* in addition to a transition model. The observation model tells us the probability that we observe $\mathbf{y}$ given that the current state is $\mathbf{x}$. In a DBN, it is convenient to represent the observation model as part of the 2TBN. We simply define a subset $\mathbf{Y} \subset \mathbf{X}$ to be the variables that are observable. We define $\mathbf{y}_t$ to be the instantiation of values for these variables observed at time t.

DBNs provide a representation language for dynamic systems that allows us to represent complex distributions in a factored way and that accomodates both discrete and continuous variables. It accomodates, as special cases, the important classes of *hidden Markov models* (HMMs) (Rabiner and Juang 1986) and *Kalman filters*. Both can be viewed as a very simple DBN with two variables X and Y, where the variable X represents the current (hidden) state of the system, and the variable Y the observation. The structure of both models is shown in Figure 4. In the case of an HMM, X is discrete, and typically so is Y. In the case of a Kalman filter, X and Y are usually vector-valued continuous variables, and the CPDs are linear Gaussians. Other, more expressive, variants of an HMM (such as the *factorial HMM* of (Ghahramani and Jordan 1997)) can also be viewed as a DBN (Smyth, Heckerman and Jordan 1996).[2]

[2] We note that a discrete DBN can also be viewed as an HMM by agglomerating all of the variables into a single "State" variable, and defining the transition model for that single variable using $p_\rightarrow$. However, the size of the resulting representation is exponential in the number of state variables, and it loses all of the regularity encoded in the DBN structure.

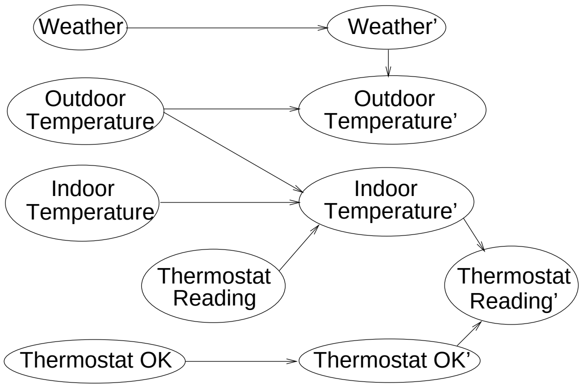


Figure 5: The thermostat network.

As an example for a 2-TBN using various types of CPDs we consider the thermostat network, shown in Figure 5. This 2-TBN describes a simple process with two phases. A thermostat measures the (continuous) temperature inside the house. It can be operational, or it can malfunction. If it is working properly it gives a reading which is a softmax classification of the room temperature into one of three readings: *Cold*, *Normal* and *Warm*. If the thermostat malfunctions, it gives one of the possible readings with a uniform distribution. The room temperature is a conditional Gaussian which depends on the previous room temperature, the outside temperature and the thermostat. If the thermostat shows *Cold*, it turns on a heating device and the room temperature is likely to go up; if the thermostat shows *Warm*, it turns on the air conditioning. Finally the outside temperature is a conditional Gaussian which depends on the weather.

3 Particle filtering for DBNs

Particle filtering is a general purpose Monte Carlo scheme for tracking in dynamic systems. It maintains the belief state at time t as a set of *particles* $\mathbf{x}_t^{(1)}, \ldots, \mathbf{x}_t^{(N_t)}$, where each $\mathbf{x}_t^{(i)}$ is a fulli nstantiation of $\mathbf{X}_t$. To generate a new (time $t+1$) particle $\mathbf{x}_{t+1}^{(j)}$ from a (time t) particle $\mathbf{x}_t^{(i)}$, we simply sample it. Ideally, we want to sample it from the distribution $p(\mathbf{X}' \mid \mathbf{X} = \mathbf{x}_t^{(i)}, \mathbf{Y}' = \mathbf{y}_{t+1})$. However, this distribution is rarely one that we can compute efficiently. Hence, we use importance sampling: we sample from one distribution, and use an importance weight to make up for the difference.

The sampling algorithm for generating a new particle and an importance weight in a DBN is roughly as follows. The algorithm traverses the network in an order which is consistent with the partial order implied by the edges. It picks a value for every variable X' in turn. A value for an unobserved variable is sampled according to the appropriate conditional distribution, given the values already selected for its parents. A variable X' whose value is observed

```
SampleParticle(x, y)
      /*x is the state at time t
        y is the observation at time t+1 */
  w := 1
  Set x' to be empty
  For i := 1 to n
    Let u be the assignment to Parents(X'_i) in x, x'
    If X'_i is not in Y'
      Sample x'_i from p(X'_i | Parents(X'_i) = u)
    Else
      Set x'_i to be X_i's observed value in y
      Set w := w · p(x'_i | Parents(X'_i) = u)
  Return(x', w)
```

Figure 6: Algorithm for sampling a particle in a DBN.

as evidence is not sampled; rather it is simply instantiated to its observed value x', and we update the weight of the new particle to reflect the probability that X *would* have been sampled as x', given the current instantiation of its parents. It is straightforward to verity that, while our algorithm only generates points $\mathbf{x}'$ that are consistent with our observations, the expected weight for any such $\mathbf{x}'$ (i.e., the probability with which it is generated times its weight when it is) is exactly its probability $p(\mathbf{x}')$. Thus, our weighted samples are an unbiased estimator of the (unnormalized) distribution over the states and the observations. The formal version of the algorithm is shown in Figure 6; we assume that $X'_1, \ldots, X'_n$ is an ordering which is consistent with the edges in the 2TBN, in that if $i < j$ then X'_j cannot be an ancestor of X'_i.

Note that there are several distinctions between this algorithm and a naive application of particle filtering. In most applications of particle filtering, it is customary to sample $\mathbf{z}'$ from the prior distribution $p(\mathbf{Z}' \mid \mathbf{x}^{(i)})$, where $\mathbf{Z}' = \mathbf{X}' - \mathbf{Y}'$, and then use $p(\mathbf{y}' \mid \mathbf{z})$ as an importance weight. In DBNs, we can do better than that in cases where the evidence is not always a leaf in the network. For example, in the Thermostat network of Figure 5, the weather is an observable that is not a leaf. The DBN sampling algorithm also samples a particle without explicitly generating the entire transition model $p(\mathbf{X}' \mid \mathbf{X})$.

This algorithm is called with a particle $\mathbf{x}_t^{(i)}$ at time t, and it generates a new particle $\mathbf{x}_{t+1}^{(j)}$ for time $t+1$, and an associated weight w. It can be used with any of the many variants of particle filtering. For example, we can use it several times to generate several samples from the same particle. If we maintain particles that have weights, we can define the weight of the new particle $\mathbf{x}'$ to be the weight of the old particle $\mathbf{x}$ times w.

At a high level, the variant of particle filtering used in our experiments is standard *bootstrap sampling*: At time t, we start out with an approximate belief state represented as a set of weighted particles $\mathbf{x}_t^{(1)}, \ldots, \mathbf{x}_t^{(N_t)}$. We bootstrap sample from the belief state: we randomly select the desired number of particles from among $\mathbf{x}_t^{(1)}, \ldots, \mathbf{x}_t^{(N_t)}$, where each $\mathbf{x}_t^{(i)}$ is selected according to its relative weight. The result is a set of *unweighted* time-t particles: we have already compensated for the relative weights of the original time-t particles by sampling each one according to its weight. Each of the new time-t particles is propagated forward using the `SampleParticle` procedure. The resulting time-$t+1$ particles, with their weights, induce the next approximate belief state, which is the starting point for the next phase.

We modify the most basic implementation of this scheme by using a simple form of density estimation. Our time t samples are necessarily very sparse. In a discrete setting, that implies that many entries in the joint distribution representing the approximate belief state will be estimated to have probability zero, even if their probability in the true belief state is positive. This type of behavior can cause significant problems, as samples at time $t+1$ are only generated based on our existing samples at time t. If the process is not very stochastic, i.e., if there are parts of the state space that only transition to other parts with very low probability, parts of the space that are not represented in our particles will not be explored. Unfortunately, the parts of the space that are not represented may have non-negligible probability; our sampling process may simply have missed them earlier, or they may be the results of trajectories that appeared unlikely in earlier time slices because of misleading evidence.

We can use simple density estimation to address this concern. In the continuous case, we simply place a small Gaussian kernel around each of our particles. In the discrete case, we are trying to estimate a density over the joint probability space of $\mathbf{X}_{t+1}$. We smooth the distribution as follows. Let $w_{t+1}^{(1)}, \ldots, w_{t+1}^{(N_{t+1})}$ be the weights of the particles generated at time $t+1$. Let $W_{t+1} = \sum_i w_{t+1}^{(i)}$. We add a certain fraction of W to each entry in the joint consistent with the evidence. In other words, our new approximate belief state — the one from which bootstrap particles will be generated — is defined as follows: for each $\mathbf{x}'$ consistent with $\mathbf{y}_{t+1}$, we define

$$\pi_t(\mathbf{x}') = \frac{1}{Z} \sum_{i \,:\, \mathbf{x}_{t+1}^{(i)} = \mathbf{x}'} w_{t+1}^{(i)} + \alpha$$

where α is some small fraction of W_t and Z is a normalizing constant. This effect is equivalent to doing Bayesian density estimation over the belief state, using a simple Dirichlet prior (DeGroot 1970). We note that, even though $\pi_t(\mathbf{x}') > 0$ for every $\mathbf{x}'$, we need only represent explicitly those states which have materialized in our sampling algorithm. The others all have the same weight α. We can easily sample from this distribution using a simple two-phase process. Let $W = \sum_i w_{t+1}^{(i)}$; then $Z = W + \alpha M$, where M is the total number of states consistent with $\mathbf{y}_{t+1}$. [3] With probability $(M\alpha)/Z$, we select some state consistent with the evidence; each such state is then selected uniformly at random. With probability W/Z, we use our standard bootstrap sampling process from the set of time-t samples. Thus, the cost of using the smoothed belief state is no higher than that of maintaining our original sparse set of samples.

The use of density estimation, and the associated smoothing effect, serves to "spread out" some of the probability mass over unobserved states, increasing the amount of exploration done for unfamiliar regions of the space.

4 Experimental results

We have tested particle filtering on two large DBNs. The first network is the discrete BAT network (Huang et al. 1994), used for monitoring freeway traffic (see Figure 7). The network describes a car driving on a freeway, by modeling the car velocity and acceleration both on the X axis (lane changes) and on the Y axis (forward movement). The network also models relations between the car and nearby cars on the freeway. The belief state of this network includes 10 variables; altogether, there are 66,528 possible states in the belief state.

Our first experiment with this network was to see how well we can track it using particle filtering. To do so, we randomly generated an observation sequence from the network. At each time step, we compared the results of running particle filtering with the correct distributions computed using exact inference (which is feasible for a model of this size).

We used two different error metrics: KL-distance (Cover and Thomas 1991) and L_1-norm. For each of these metrics, we computed the error on the entire joint distribution of the belief state and also the average error on the marginals of all the variables in the belief state. Obviously, estimating the marginals is much easier than estimating the full joint distribution, therefore the average error over the marginals is much smaller.

Figure 8 shows the error as a function of the number of samples. In both cases we report the average error over the entire run. The results were averaged over 10 runs. Figure 8(a) shows the KL error on the entire belief state. Figure 8(b) shows the average L1-norm error over the marginals of the random variables. As expected, the error drops sharply initially and then the improvements become smaller and smaller. Note that the dropoff occurs at around 8000 samples, substantially less than the number of states in the belief state.

[3] M can easily be computed when all CPDs do not contain zero entries. If CPDs do contain zero entries, the situation is more problematic. In general, deciding which instantiations are consistent with the evidence is an NP-hard problem, so we either have to resign ourselves to having a small positive probability even for some states that are inconsistent with our current evidence, or we must resort to some other smoothing scheme.

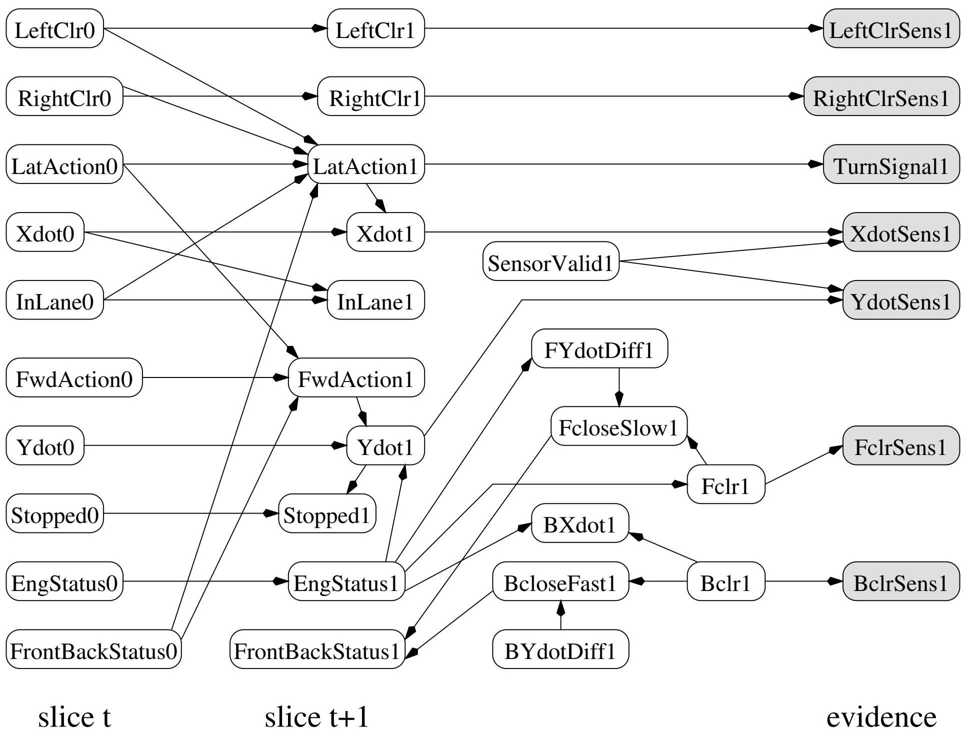
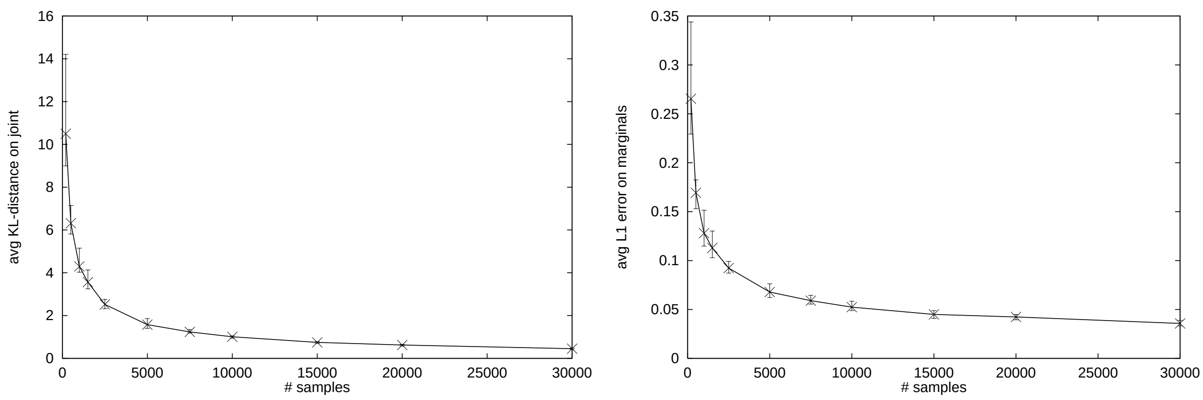


Figure 7: The BAT network.



(a) (b)

Figure 8: Error as a function of the number of samples

Figure 9 shows how the errors behaves over the entire run. We present the KL-error, averaged over 10 runs with identical evidence (L1-norm has a similar behavior) with 5000 samples per time step. It is interesting to see that the error changes dramatically over the sequence. Generally speaking, the spikes in the error correspond to unlikely evidence, in which case the samples become less reliable. This phenomenon suggests that it may be advantageous to use a more adaptive sampling scheme, e.g., by using a different number of samples at different time steps.

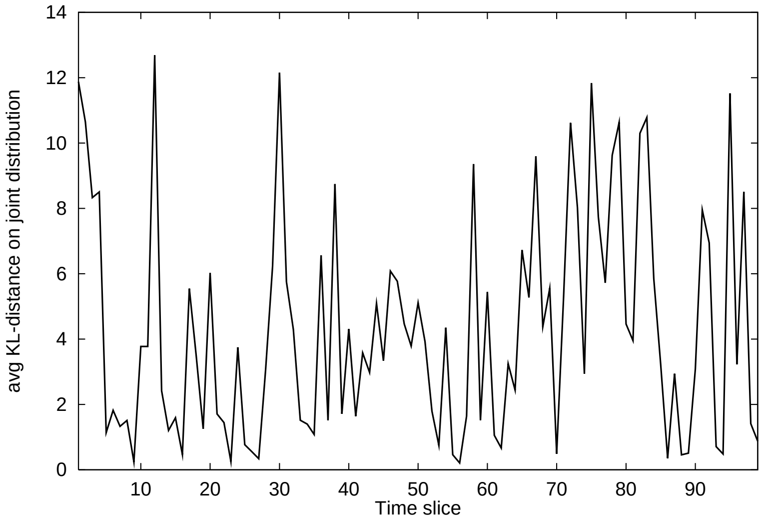


Figure 9: The KL error over a sequence, averaged over 10 runs

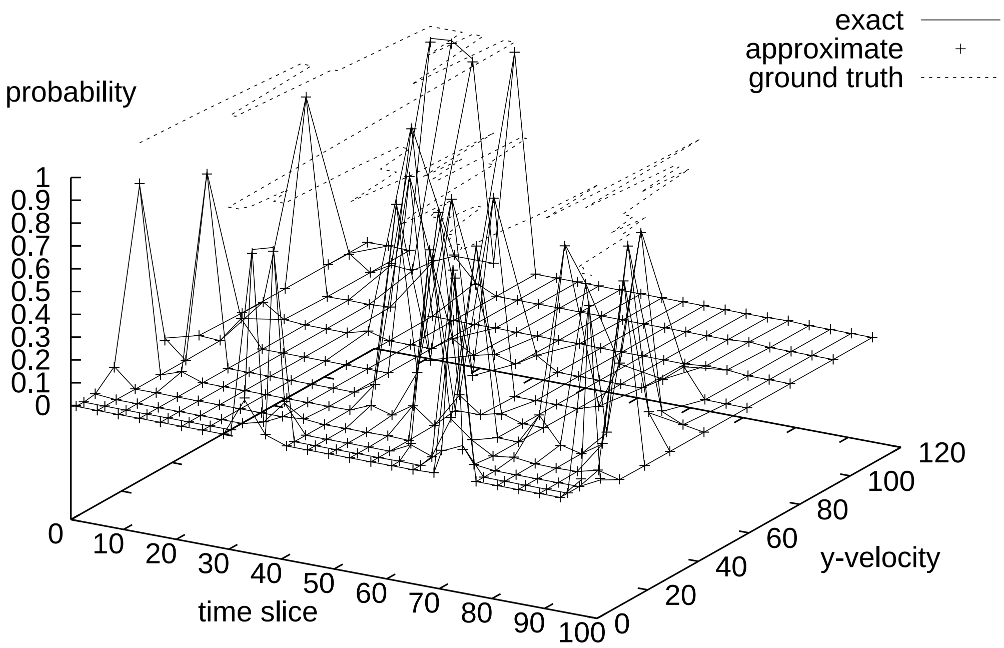


Figure 10: The exact and approximate belief states for the *Ydot1* variable, and the ground truth used to generate the trajectory.

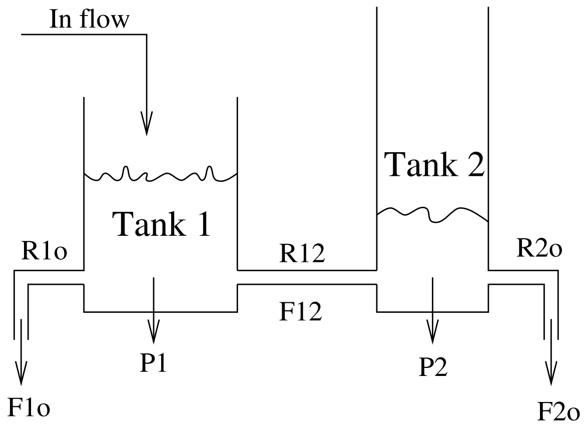


Figure 11: The Two Water Tanks System.

To further investigate the quality of the approximation we focused on one of the variables in the belief state: the forward velocity (*Ydot1* in Figure 7). In this case we used 5000 samples per time step. The results are given in Figure 10. As can be seen, particle filtering gives a very good estimate of the true distribution over this variable. We note that the error for any particular value of the variable is tiny; the error of 0.06 suggested by Figure 9 is the result of summing over all possible values of the variable.

The second network we experimented with is a hybrid network that models a process which is commonly used as a benchmark in the fault diagnostics community (Mosterman and Biswas 1999). The physical system is shown in figure Figure 11 and the DBN is shown in Figure 12.

The system contains two water tanks which are connected by a pipe. Each one of the tanks has a pipe from which water flow out of the tank. The first tank also has a constant flow of water going into it. We get noisy measurements of the flows in each one of the three pipes.

There are three possible types of failures in the system:

- **Measurement failures:** Usually measurements are quite reliable, but in case of a measurement failure, the measurement becomes extremely noisy. In this case the value of the measurement has almost nothing to do with the actual flow; thus, to track the system correctly, it is necessary to identify these faulty measurements and ignore them.
- **Pipe bursts:** A pipe can suddenly change its resistance to some unknown value.
- **Drifts:** The resistance of the pipe can drift, which gradually increases or decreases the pipe's resistance.

The probability of a measurement failure is 0.01. The probability of a burst is 0.003 and the probability of a drift (positive or negative) is also 0.003. Once a drift begins, it persists with probability 0.991.

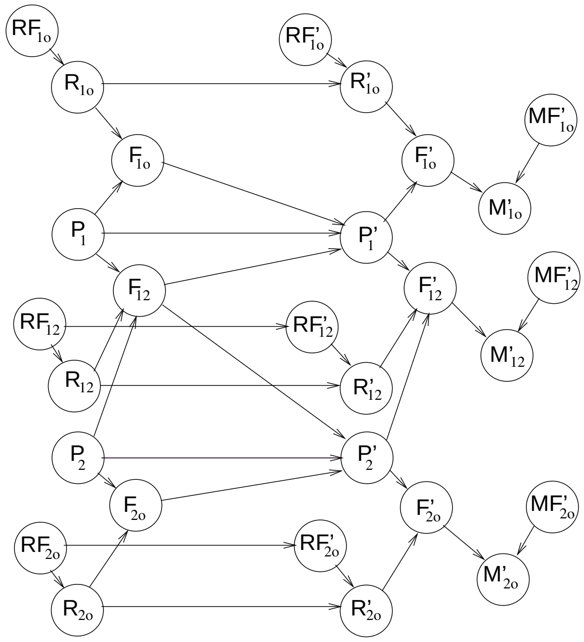


Figure 12: The *two-tank* DBN.

The DBN which models the two-tank system is shown in Figure 12. The nodes labeled by RF indicate faults in the resistances of the pipes (drifts or bursts) and the nodes labeled by MF indicate measurement failures. The nodes labeled P, F, and R are continuous and indicate pressures, flows, and pipe resistances, respectively.

By explicitly modeling the pipe resistances, we accurately model the physical system. However, since the flow is the ratio between the pressure and the resistance, the system is not a linear system. Dealing with ratios is difficult, especially when the values are close to zero. Therefore instead of modeling the resistances we choose to model the conductances (the conductance is defined as the reciprocal of the resistance). This transformation results in products rather than ratios. The system is still nonlinear, but this does not pose a problem for the particle filtering algorithm, which is general enough to deal with nonlinear CPDs.

Note the difference between the measurement failure nodes and the pipe faults. The pipe faults are persistent, and therefore appear both at time t and at time $t+1$. In fact, the pipe faults are a part of the belief state. Measurement failures, on the other hand, are transient and therefore appear only at time $t+1$ and need not be included in the belief state. As a result, the network has six pipe fault variables and three measurement failure variables, leading to 32,768 different discrete states.

Unfortunately, this network is far too complicated for us to be able to use exact inference. In fact, the belief state at time t is a mixture of Gaussians, with a number of mixture components that grows exponentially with t, and

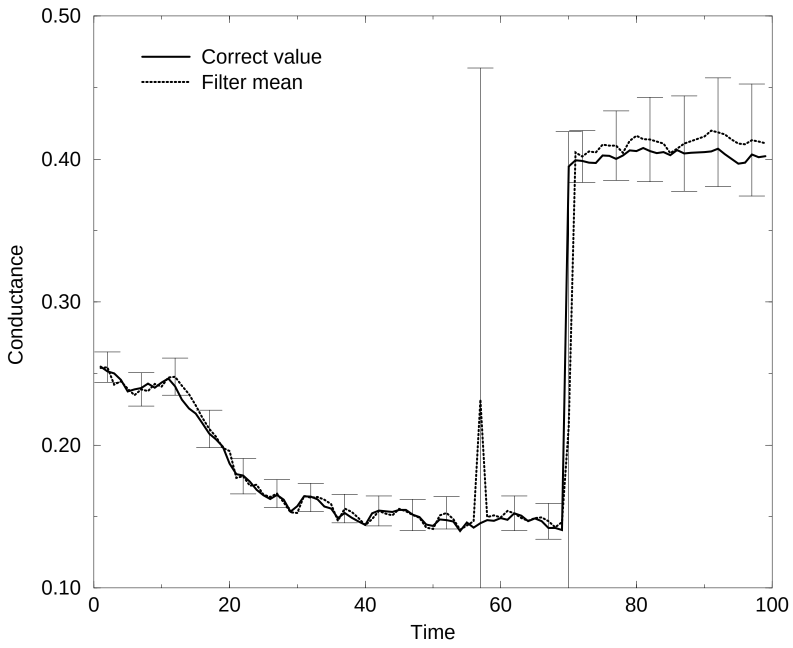


Figure 13: Tracking R_{1o}

with the number of discrete variables in each time slice. Hence, we examine how well the algorithm tracks the value of one particular random variable in a complex sequence of failures.

Our sequence includes the following failures:

- At $t = 12$ a negative drift begins in R_{1o} (the output pipe of the first tank).
- At $t = 23$, R_{12} bursts.
- At $t = 31$ we have a measurement failure for F_{12}.
- At $t = 42$ the negative drift for R_{1o} ends.
- At $t = 57$ we have another measurement failure, this time for F_{1o}.
- At $t = 70$ and at $t = 81$ we have bursts at R_{1o} and R_{2o}, respectively.

The most interesting variable to track during this sequence is R_{1o} which experienced all types of failures: a drift, a measurement failure and then a burst. The results, based on 10 runs with 50000 samples in each run, are shown in Figure 13. The results show the true value of R_{1o}, the mean of the samples for R_{1o} and the confidence interval (taken to be plus or minus two standard deviations from the mean).

We can see that we are able to track the resistance quite well. The algorithm correctly finds the negative drift and tracks the resistance accordingly. The probability that the algorithm assigns to a negative drift increases very gradually, as it takes quite a bit of evidence to support this unlikely event. When the drift begins at $t = 12$ the probability of a negative drift is only 0.008. This increases to 0.145 at $t = 16$, 0.86 at $t = 21$ and 0.993 at $t = 25$. When the drift ends at $t = 42$ the probability gradually goes down.

The big increase in uncertainty at $t = 57$ is due to the measurement error. At this point the algorithm considers two hypotheses — either a burst or a measurement error. As more evidence is collected, the hypothesis of a measurement failure dominates, and the algorithm again tracks the resistance correctly. At $t = 70$ there is a similar increase in uncertainty, but this time the burst hypothesis is consistent with the subsequent evidence, and the algorithm continue to track the resistance correctly (although with a somewhat bigger uncertainty over the true value).

5 Conclusions

Dynamic Bayesian networks are a rich modeling language that allows us to represent very complex stochastic systems, including ones that involve both discrete and continuous variables. Particle filtering provide a general-purpose inference algorithm that can be applied to virtually any DBN (we only require that the representation of the CPDs admits sampling). Thus, it allows us to deal, at least in principle, with an extremely rich class of dynamic systems.

The ability to represent very large-scale systems raises the question of whether particle filtering approaches scale up to very high-dimensional spaces. In this paper, we have examined this issue in the context of two realistic systems — one discrete and one hybrid. Our results indicate that particle filters are surprisingly robust, even for complex systems. Nevertheless, we feel that the curse of dimensionality that plagues instance-based methods is also an issue for particle filtering in high-dimensional spaces. Therefore, we are currently exploring approaches that address this concern. Two ideas seem particularly useful in this setting:

- Combinations of sampling with exact inference — an extension of Rao-Blackwellization (Doucet, Godsill and Andrieu 2000); this idea was applied successfully to a robot localization and mapping task in (Murphy and Russell 2000).
- Approximate belief state representations that exploit the weak interaction in the domain structure, as was done in (Boyen and Koller 1998, Boyen and Koller 1999) for discrete DBNs using non-sampling-based methods.

We believe that a combination of these techniques will provide us with the tools to reason about the kinds of complex hybrid systems encountered in many real-world applications.

Acknowledgements

We would like to thank Dragomir Angelov, Xavier Boyen, and Raya Fratkina for their help on this project. This research was supported by ARO under the MURI program "Integrated Approach to Intelligent Systems", grant number DAAH04-96-1-0341, by an ONR Young Investigator Award grant number N00014-99-1-0464, by the Powell Foundation and by the Sloan Foundation.

References

Bar-Shalom, Y. and Fortmann, T. E. (1988). *Tracking and Data Association*, Academic Press.

Bishop, C. M. (1995). *Neural networks for pattern recognition*, Oxford University Press.

Boyen, X. and Koller, D. (1998). Tractable inference for complex stochastic processes, *Proc. Fourteenth Annual Conference on Uncertainty in AI (UAI)*, pp. 33–42.

Boyen, X. and Koller, D. (1999). Exploiting the architecture of dynamic systems, *Proc. Sixteenth National Conference on Artificial Intelligence (AAAI)*, pp. 313–320.

Cover, T. and Thomas, J. (1991). *Elements of Information Theory*, Wiley.

Dagum, P. and Galper, A. (1995). Forecasting sleep apnea with dynamic network models, *Proc. Ninth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 64–71.

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

DeGroot, M. (1970). *Optimal Statistical Decisions*, McGraw-Hill, New York, NY.

Doucet, A., Godsill, S. and Andrieu, C. (2000). On sequential Monte Carlo sampling methods for Bayesian filtering, *Statistics and Computing*.

Fox, D., Burgard, W. and Thrun, S. (1999). Markov localization for mobile robots in dynamic environments, *Journal of Artificial Intelligence Research (JAIR)* **11**: 391–427.

Ghahramani, Z. and Jordan, M. I. (1997). Factorial hidden Markov models, *Machine Learning* **29**(2–3): 245–273.

Gilks, W., Richardson, S. and Spiegelhalter, D. (1996). *Markov Chain Monte Carlo Methods in Practice*, CRC Press.

Gordon, N. J., Salmond, D. J. and Smith, A. F. M. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation, *IEE Proceedings-F* **140**(2): 107–113.

Handschin, J. E. and Mayne, D. Q. (1969). Monte Carlo techniques to estimate the conditional expectation in multi-stage non-linear filtering, *International Journal of Control* **9**(5): 547–559.

Huang, T., Koller, D., Malik, J., Ogasawara, G., Rao, B., Russell, S. J. and Weber, J. (1994). Automatic symbolic traffic scene analysis using belief networks, *Proc. Twelfth National Conference on Artificial Intelligence (AAAI)*, pp. 966–972.

Isard, M. and Blake, A. (1998). CONDENSATION — conditional density propagation for visual tracking, *International Journal of Computer Vision* **29**(1): 5–28.

Jensen, F., Kjærulff, U., Olesen, K. and Pedersen, J. (1989). An expert system for control of waste water treatment— a pilot project, *Technical report*, Judex Datasystemer A/S, Aalborg, Denmark.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems, *Transactions ASME, Journal of Basic Engineering* **82**: 34–45.

Kanazawa, K., Koller, D. and Russell, S. (1995). Stochastic simulation algorithms for dynamic probabilistic networks, *Proc. Eleventh Annual Conference on Uncertainty in AI (UAI '95)*, pp. 346–351.

Koller, D., Lerner, U. and Angelov, D. (1999). A general algorithm for approximate inference and its application to hybrid Bayes nets, *Proc. Fifteenth Annual Conference on Uncertainty in AI (UAI)*, pp. 324–333.

Mosterman, P. and Biswas, G. (1999). Diagnosis of continuous valued systems in transient operating regions, *IEEE Transactions on Systems, Man, and Cybernetics, Part A* **29**(6): 554–565.

Murphy, K. and Russell, S. (2000). title, *This collection*.

Nicholson, A. E. and Brady, J. M. (1994). Dynamic belief networks for discrete monitoring, *IEEE Trans. Systems, Man and Cybernetics* **24**(11): 1593–1610.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann.

Rabiner, L. and Juang, B. (1986). An introduction to hidden Markov models, *IEEE Acoustics, Speech & Signal Processing.*

Shachter, R. and Kenley, C. (1989). Gaussian influence diagrams, *Management Science* **35**: 527–550.

Smyth, P., Heckerman, D. and Jordan, M. (1996). Probabilistic independence networks for hidden Markov probability models, *Neural Computation.*

Zweig, G. and Russell, S. J. (1998). Speech recognition with dynamic Bayesian networks, *Proc. Fifteenth National Conference on Artificial Intelligence (AAAI)*, pp. 173–180.