

Using learning for approximation in stochastic processes

Daphne Koller

Computer Science Dept.
Stanford University
Stanford, CA 94305-9010
koller@cs.stanford.edu

Raya Fratkina

Computer Science Dept.
Stanford University
Stanford, CA 94305-9010
raya@cs.stanford.edu

Abstract

To monitor or control a stochastic dynamic system, we need to reason about its current state. Exact inference for this task requires that we maintain a complete joint probability distribution over the possible states, an impossible requirement for most processes. Stochastic simulation algorithms provide an alternative solution by approximating the distribution at time t via a (relatively small) set of samples. The time t samples are used as the basis for generating the samples at time $t + 1$. However, since only existing samples are used as the basis for the next sampling phase, new parts of the space are never explored. We propose an approach whereby we try to generalize from the time t samples to unsampled regions of the state space. Thus, these samples are used as data for learning a distribution over the states at time t , which is then used to generate the time $t + 1$ samples. We examine different representations for a distribution, including density trees, Bayesian networks, and tree-structured Bayesian networks, and evaluate their appropriateness to the task. The machine learning perspective allows us to examine issues such as the tradeoffs of using more complex models, and to utilize important techniques such as regularization and priors. We validate the performance of our algorithm on both artificial and real domains, and show significant improvement in accuracy over the existing approach.

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 ICU, analyzing a complex freeway traffic scene with the goal of controlling a moving vehicle, or even tracking motion of objects in a visual scene. Such systems have complex and unpredictable dynamics; thus, they are often modeled as stochastic dynamic systems. Even when a model of

the system is known, reasoning about the system is a computationally difficult task. Our main concern in this paper is in using machine learning techniques as part of a reasoning task; specifically, the task of *monitoring* the state of the system as it evolves and as new observations are obtained.

Theoretically, the monitoring task is straightforward. We simply maintain a probability distribution over the possible states at the current time. As time evolves, we update this distribution using the transition model; as new observations are obtained, we use Bayesian conditioning to update it. Such a distribution is called a *belief state*; in a Markovian process, it provides a concise summary of all of our past observations, and suffices both for predicting the future trajectory of the system as well as for making optimal decisions about our actions [Ast65].

Unfortunately, even systems whose evolution model is compactly represented rarely admit a compact representation of the belief state and an effective update process. Consider, for example, a stochastic system represented as a *dynamic Bayesian network (DBN)* [DK89]. A DBN partitions the evolution of the process into *time slices*, each of which represents a snapshot of the state of the system at one point in time. Like a Bayesian network (BN), the DBN utilizes a decomposed representation of the state via state variables and a graphical notation to represent the direct dependencies between the variables in the model. The evolution model of the system—the distribution over states at time $t + 1$ given the state at time t —is represented in a network fragment such as the one in Figure 1(a) (appropriately annotated with probabilities). DBNs have been used for a variety of applications, including freeway surveillance [FHKR95], monitoring complex factories [JKOP89], and more.

Exact inference algorithms for BNs have analogues for inference in DBNs [Kja92]. Unfortunately, in most cases, these algorithms also end up maintaining a belief state—a distribution over most or all of the variables in a time slice. Furthermore, it can be shown [BK98] that the belief state rarely has any structure that may support a compact representation. Thus, exact inference algorithms are forced to maintain a fully explicit joint distribution over an exponen-

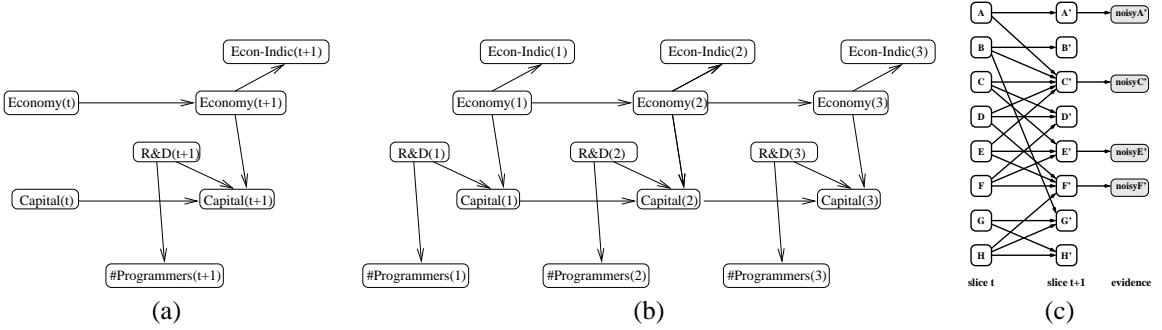


Figure 1: (a) The simple CAPITAL 2TBN for tracking the growth of a hi-tech company; (b) The same 2TBN unrolled for 3 time slices; (c) The WATER 2TBN.

tially large state space, making them impractical for most complex systems.

A similar problem arises when we attempt to monitor a process with complex continuous dynamics. Here also, an explicit representation of the belief state is infeasible.

This limitation has led to work on approximate inference algorithms for complex stochastic processes [GJ96, BK98, KKR95, IB96]. Of the approaches proposed, stochastic simulation algorithms are conceptually simplest and make the fewest assumptions about the structure of the process. The *survival of the fittest (SOF)* algorithm [KKR95] has been applied with success to large discrete DBNs [FHKR95]. The same algorithm (independently discovered by [IB96]) has been applied to the continuous problem of tracking object motion in cluttered visual scenes.

The algorithm, which builds on stochastic simulation algorithms for standard BNs [SP89], is as follows: For each time slice, we maintain a (small) set of weighted samples; a sample is one possible state of the system at that time, while its weight is some measure of how likely it is. This set of weighted samples is, in effect, a very sparse estimate of the belief state at time t . A sample at time t is propagated to time $t + 1$ by a random process based on the dynamics of the system. In a naive generalization of [SP89], each time t sample is propagated forward to time $t + 1$. However, as shown by [KKR95], this approach results in extremely poor performance, with the error of the approximation diverging rapidly as t grows. They propose an approach where samples are propagated preferentially: those whose weight is higher are more likely to be propagated, while the lower weight ones tend to be “killed off.” Technically, samples from time t are selected for propagation using a random process that chooses each sample proportionately to its weight. The resulting trajectories are weighted based on how well they fit the new evidence at time $t + 1$, and the process continues. Despite its simplicity and low computational cost, the SOF algorithm performs very well; as shown in [KKR95], its error seems to remain bounded indefinitely over time. As shown in [IB96], this algorithm can also deal with complex continuous processes

much more successfully than standard techniques.

In this paper, we use machine learning techniques to improve the behavior of the SOF algorithm, with the goal of applying it to real-world complex domains. The SOF algorithm shifts its effort from less likely to more likely trajectories, thereby focusing on the more relevant parts of the space. However, at time $t + 1$, it only samples parts of the space that arise from samples that it had at time t . Thus, it does not allow for correcting earlier mistakes: if its samples at time t were unrepresentative in some way, then its samples at time $t + 1$ are also likely to be so. We can reinterpret this behavior from a somewhat different perspective. The set of time t samples are an approximation to the belief state at time t . When SOF chooses which samples to propagate it is simply sampling from this approximate belief state. Thus, the SOF algorithm is using a set of weighted samples as an approximation to a belief state, and a random sampling process to propagate an approximate belief state at time t to one at time $t + 1$.

This perspective is a natural starting point for our approach. Clearly, a small number of weighted points is a suboptimal way of representing a complex distribution over a large space: As the number of samples is much smaller than the total size of the space, the representation is very sparse, and therefore necessarily unrepresentative. Our key insight is that our information about the relative likelihood of even a small number of points in the space can tell us a lot about the relative likelihood of others. Thus, we can treat our samples as data cases and use them to *learn* the shape of the distribution. In other words, we can use our own randomly generated samples as input to a density estimation algorithm, and use them to learn the distribution.

This insight leads us to explore a number of improvements to the SOF algorithm. We first note, in Section 3, that the number of samples needed to adequately estimate the distribution can vary widely: in situations where the evidence is unlikely, more samples will be needed in order to “find” relevant regions of the space. Luckily, as we are generating our own data, we can generate as many samples as we need; that is, we can perform a simple type of *active learning* [CAL94].

We then introduce a Dirichlet prior over the parameters of our distribution in order to deal with the problem of numerical overfitting, a particularly serious problem when we have a sparse sample for a very large space. We show that even these two simple improvements serve to significantly increase the accuracy of our algorithm.

We then proceed to investigate the issue of generalizing from the samples to other parts of the space. Of course, in order to generalize, we need a representation whose bias is higher. The requirements of our task impose several constraints both on the representation of the distribution and on the algorithm used to estimate it. First, as our state space is exponentially large, we must restrict attention to compact representations of distributions. Second, we must allow samples to be generated randomly from the distribution in a very efficient way. Thus, for example, a neural network whose input is a possible state of the process and whose output is the probability of that state would not be appropriate. Finally, as we are primarily interested in fast monitoring in time-critical applications, we prefer density estimation algorithms that are less compute-intensive.

Based on these constraints, we explore three main approaches, appropriate to processes represented as DBNs: Bayesian networks with a fixed structure, tree-structured Bayesian networks with a variable structure, and *density trees*, which resemble decision trees or discrete regression trees. We compare the performance of these algorithms to that of the SOF algorithm, and show that all three track the process with higher accuracy. We show that the density tree approach seems particularly promising, and suggest a possible explanation as to why it behaves better than the other approaches. We conclude with some discussion and possible extensions of our approach to other domains.

2 Preliminaries

A discrete time stochastic process is viewed as evolving randomly from one state to another at discrete time points. Formally, there is a set of states Σ such that at any point in time t , the situation can be described using some state $\mathbf{x} \in \Sigma$. 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 . Formally, letting $\mathbf{X}^{(t)}$ denote the random variable (or set of random variables) representing the state at time t , we have that $\mathbf{X}^{(t+1)}$ is independent of $\mathbf{X}^{(0)}, \dots, \mathbf{X}^{(t-1)}$ given $\mathbf{X}^{(t)}$. Thus, $P(\mathbf{X}^{(0)}, \dots, \mathbf{X}^{(t)}) = P(\mathbf{X}^{(0)}) \prod_{l=1}^t P(\mathbf{X}^{(l)} | \mathbf{X}^{(l-1)})$. We also typically assume that the process is time invariant, so that $P(\mathbf{X}^{(t)} | \mathbf{X}^{(t-1)})$ 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 time t is specified in terms of a set of state variables $X_1^{(t)}, \dots, X_n^{(t)}$. The transition model therefore has to define a probability distribution $P(X_1^{(t+1)}, \dots, X_n^{(t+1)} | X_1^{(t)}, \dots, X_n^{(t)})$. We specify such

a distribution using a network fragment called a *2TBN*—a two time-slice Bayesian network, as shown in Figure 1(a). A 2TBN defines the probability distribution for any time slice $t+1$ given time slice t : for each variable $X_i^{(t+1)}$ in the second time slice, the network fragment specifies a set of parents $Parents(X_i^{(t+1)})$, which can be variables either in time slice $t+1$ or in time slice t ; it also specifies a conditional probability table, which describes the probability distribution over the values of $X_i^{(t+1)}$ given any possible combination of values for its parents. As a whole, the 2TBN completely specifies $P(\mathbf{X}^{(t+1)} | \mathbf{X}^{(t)})$. The network fragment can be *unrolled* to define a distribution over arbitrarily many time slices. Figure 1(b) shows the 2TBN of Figure 1(a) unrolled over three time slices.

The state of the process is almost never fully observable; thus, in any time slice, we will get to observe the values of only some subset of the variables. In most monitoring tasks, the set of observable variables is the same in the different time slices. These variables typically represent sensor readings, e.g., the reading of some blood-pressure monitor in the ICU or the output of a video camera on a freeway overpass. Let $O^{(t)}$ be the set of observable variables at time t , and let $\mathbf{o}^{(t)}$ be the instantiation of values for these variables observed at time t . In the *monitoring* task, we are interested in reasoning about $X_1^{(t)}, \dots, X_n^{(t)}$ given all the observations seen so far; i.e., we want to maintain $P(\mathbf{X}^{(t)} | \mathbf{o}^{(0)}, \dots, \mathbf{o}^{(t)})$.

As we discussed in the introduction, the stochastic simulation algorithms for DBNs is based on the standard likelihood weighting (LW) algorithm. The algorithm, shown in Figure 2, generates a sample by starting at the roots of the network and continuing in a top-down fashion, picking a value for every variable in turn. A value for a variable is sampled according to the appropriate conditional distribution, given the values already selected for its parents. Variables whose values were observed as evidence are not sampled; rather the variable is simply instantiated to its observed value. However, we must compensate for the fact that we forced this variable to take a value which may or may not be likely. Thus, we modify the weight of the sample to reflect the likelihood of having observed this particular value for this variable. It is easy to see 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. Thus, our weighted samples are an unbiased estimator of the (unnormalized) distribution over the states and the observations. Note that the weight of the sample represents how well it explains our observations. Thus, a sample of very low weight is a very bad explanation for the observations, and contributes very little to our understanding of the situation.

The straightforward application of LW to DBNs is simply by treating the DBN as one very long BN. Roughly

```

LikelihoodWeighting( $\mathbf{x}^{(t)}, \mathbf{o}^{(t+1)}$ )
   $w := 1$ 
  for  $i := 1$  to  $n$ 
    Let  $\mathbf{u}$  be the assignment to  $Parents(X_i^{(t+1)})$  in  $\mathbf{x}^{(t)}, \mathbf{x}^{(t+1)}$ 
    If  $X_i^{(t+1)}$  is not in  $\mathbf{O}^{(t+1)}$ 
      Sample  $x_i^{(t+1)}$  from  $P(X_i^{(t+1)} | Parents(X_i^{(t+1)}) = \mathbf{u})$ 
    Else
      Set  $x_i^{(t+1)}$  to be  $X_i$ 's observed value in  $\mathbf{o}^{(t+1)}$ 
    Set  $w := w \cdot P(X_i^{(t+1)} = x_i^{(t+1)} | Parents(X_i^{(t+1)}) = \mathbf{u})$ 
  Return( $\mathbf{x}^{(t+1)}, w$ )

```

Figure 2: A temporal version of the likelihood weighting algorithm; it generates an instantiation $\mathbf{x}^{(t+1)}$ for the time $t + 1$ variables given an instantiation $\mathbf{x}^{(t)}$ for the time t variables.

speaking, the algorithm would maintain a set of samples $\mathbf{x}^{(t)}[1], \dots, \mathbf{x}^{(t)}[N]$ for every time slice t , representing possible states of the process at that time. At each time slice t , each of the samples is propagated to the next time slice using the LW algorithm, and its weight is adjusted according to how well it reflects the new observations. Unfortunately, as observed by [KKR95], this approach works very poorly for most DBNs. Intuitively, the process by which samples are randomly generated is oblivious to the evidence, which only affects the weight assigned to the samples. Therefore, the samples represent random trajectories through the system, most of which are completely irrelevant. As a consequence, as shown in [KKR95], the accuracy of LW diverges extremely quickly over time.

The *survival of the fittest* algorithm of [KKR95] addresses this problem by preferentially selecting which samples to propagate according to how likely they are, i.e., their weight relative to other samples. Technically, each sample $\mathbf{x}^{(t)}[j]$ is associated with a weight $w^{(t)}[j]$. In order to propagate to the next time slice, the algorithm first renormalizes all of the weights $w^{(t)}[j]$ to sum to 1. Then, it generates N new samples for time $t + 1$ as follows: For each new sample j , it selects $\mathbf{x}^{(t)}$ randomly from among $\mathbf{x}^{(t)}[1], \dots, \mathbf{x}^{(t)}[N]$, according to their weight. It then calls LW with $\mathbf{x}^{(t)}$ as a starting point, and gets back a time $t + 1$ sample $\mathbf{x}^{(t+1)}$ and a weight w . It sets $\mathbf{x}^{(t+1)}[j] := \mathbf{x}^{(t+1)}$ and $w^{(t+1)}[j] := w$. Note that the weight of the sample $w^{(t)}[j]$ manifests in the relative proportion with which $\mathbf{x}^{(t)}[j]$ will be propagated, so that we do not need to recount it when defining $w^{(t+1)}[j]$. Kanazawa *et al.* show empirically that, unlike LW, the error of SOF seems to remains bounded indefinitely over time.

3 Belief state estimation

We can interpret the SOF algorithm as estimating a probability distribution over the states at time t . Having generated some number of samples $\mathbf{x}^{(t)}[1], \dots, \mathbf{x}^{(t)}[N]$, it renormalizes their weights to sum to 1. The result is a simple

count distribution $\sigma_{sc}^{(t)}$ over the states at time t , one which gives some positive probability to states that correspond to one or more samples, and zero probability to all the rest. The SOF algorithm then generates N new samples from $\sigma_{sc}^{(t)}$, and propagates each of them to time $t + 1$ using the LW algorithm. The result samples are again renormalized, and the process repeats.

The distribution $\sigma_{sc}^{(t)}$ is a compact approximation to the belief state $\sigma^{(t)}$ at time t —the correct distribution $P(\mathbf{X}^{(t)} | \mathbf{o}^{(0)}, \dots, \mathbf{o}^{(t)})$. Assuming we know the initial state at time 0, $\sigma_{sc}^{(0)}$ is precisely the belief state at time 0. The properties of LW imply that our weighted samples at time 1 are an unbiased estimator for $P(\mathbf{X}^{(1)}, \mathbf{o}^{(1)} | \mathbf{o}^{(0)})$. Thus, after renormalization, $\sigma_{sc}^{(1)}$ is an estimator (albeit a biased one) for $P(\mathbf{X}^{(1)} | \mathbf{o}^{(0)}, \mathbf{o}^{(1)})$. By similar reasoning, we have that $\sigma_{sc}^{(t)}$ is a (biased) estimator for $P(\mathbf{X}^{(t)} | \mathbf{o}^{(0)}, \dots, \mathbf{o}^{(t)})$. However, each $\sigma_{sc}^{(t)}$ is only a very sparse approximation to $\sigma^{(t)}$, and thus one which is less than representative. It is also highly variable, with a strong dependence on which samples we happened to pick at the multiple previous random sampling stages. Both the sparsity and variability of our estimate propagate to the next time slice, increasing the variance of our approximation.

Our first attempt to control this variance relates to the amount of data on which our estimation is based. Naively, it seems that, at each phase, we are basing our estimation procedure on the same number N of samples. However, when we are renormalizing our distribution, we are not dividing by N , but rather by the total weight of the samples. Intuitively, if the evidence observed at a given time point is unlikely, each sample generated will explain it less well, so that its weight will be low. Thus, if the total weight of our N samples is low, then we have not really sampled a significant portion of the probability mass. Indeed, as argued by Dagum and Luby [DL97], the actual number of *effective* samples is their total weight. Thus, we modify the algorithm to guarantee that our estimation is based on a fixed weight rather than a fixed number of samples.

Our results for this improvement, applied to the simple CAPITAL network, are shown in Figure 3. The data were generated over 25 runs. In each run, the observations were generated randomly, from the correct distribution; thus they correspond to typical runs of the algorithm over a typical evidence sequence. Figure 3(a) shows the number of samples used over different time slices; we see that the number of samples varies significantly over time, illustrating that the algorithm is taking advantage of the additional flexibility. The average number of samples per time slice used over the run is 65. Figure 3(b) compares the accuracy of this algorithm to that of a fixed-samples algorithm using 65 samples in *each* time slice. We see that while the average number of samples used is the same, the variable-samples approach obtains consistently higher accuracy; in order to obtain comparable accuracy from the fixed-samples algo-

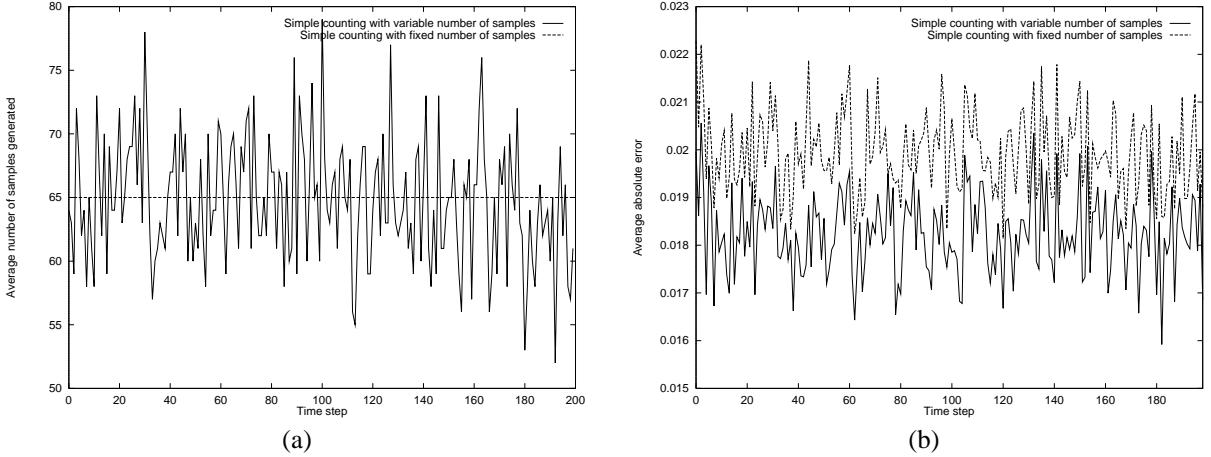


Figure 3: Comparison of variable-samples and fixed-samples algorithms for the CAPITAL network, averaged over 25 sequences: (a) number of samples used; (b) L_1 error.

rithm, around 70 samples are needed. We note that while both the error and the number of samples varies widely, they remain bounded indefinitely over time. This boundedness property continues to hold even in long runs with thousands of time slices. We also note that the number of states in the explicit belief state representation is 256, as compared to 55–80 samples used; thus our sampling approach allows considerable savings.

We have experimented with the number of samples required for different evidence sequences. Our results show that unlikely evidence sequences require many more samples than likely evidence, thereby justifying our intuition about the reason for the variability in the number of samples needed. Furthermore, the accuracy maintained by the variable-samples algorithm for likely and unlikely runs is essentially the same; thus, in a way, the algorithm generates as many samples as it needs to maintain a certain level of performance. We can view this ability as a type of *active learning* [CAL94], where the learning algorithm has the ability to ask for more data cases when necessary. In our context, the active learning paradigm is particularly appropriate, as the algorithm is generating its own data cases.

Our next improvement relates to another problem with the SOF algorithm. Our time t samples are necessarily very sparse, so that many entries in the probability distribution $\sigma_{sc}^{(t)}$ will have zero probability, even though their true probability 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 $\sigma_{sc}^{(t)}$ will not be explored. Unfortunately, the parts of the space that are not represented may be quite likely; 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. This problem is reflected clearly if we measure the distance between our approximation and the exact distribution using *relative entropy* [CT91], for many reasons the most appropriate distance measure for this type of situation. For an exact distribution ϕ and an approximate one ψ over the same space Ω , the relative entropy $D(\phi\|\psi)$ is defined as $\sum_{\omega \in \Omega} \phi(\omega) \log(\phi(\omega)/\psi(\omega))$. In cases, such as ours, where the approximate distribution ascribes probability 0 to entries that are not impossible, the relative entropy distance is infinite.

The machine learning perspective offers us a simple and theoretically well-founded solution to the problem of unwarranted zeros in our estimated distribution. We view the problem from the perspective of Bayesian learning, where we have a prior distribution over the parameters we are trying to estimate: the probabilities $\theta_{\mathbf{x}}$ of the different states \mathbf{x} in our belief state. An appropriate prior for multinomial distributions such as this is the *Dirichlet distribution*. We omit the formal definition of the Dirichlet prior, referring the reader to [Deg86]. Intuitively, it is defined using a set of hyperparameters $\alpha_{\mathbf{x}_i}$, each representing “imaginary” samples observed for the state \mathbf{x}_i . In our case, as we have no beliefs in favor of one state \mathbf{x} over another, we chose $\alpha_{\mathbf{x}_i}$ to be uniformly α/r , where r is the total number of states consistent with our evidence.

Computing with this seemingly complex two-level distribution is actually quite simple, as most computations are equivalent to working with a single distribution $\sigma_{sc+}^{(t)}$, obtained from taking the *expectation* of the parameters $\{\theta_{\mathbf{x}}\}$ relative to their prior distribution. In our case, for each \mathbf{x} consistent with our evidence, we have that $\sigma_{sc+}^{(t)}(\mathbf{x}) = (w^{(t)}(\mathbf{x}) + \alpha/r)/Z$ where $w^{(t)}(\mathbf{x})$ is the total weight of samples $\mathbf{x}^{(t)}[j]$ whose value is \mathbf{x} , and Z is a normalizing factor. We see that each instantiation \mathbf{x} in our distribution $\sigma_{sc+}^{(t)}$ (if consistent with our evidence) will have at

least some very small probability mass. We note that, even though $\sigma_{sc+}^{(t)}(\mathbf{x}) > 0$ for every \mathbf{x} , we need only represent explicitly those states which have materialized in our sampling algorithm. Thus, the cost of maintaining such a distribution is no higher than that of maintaining our original sparse set of samples.

The introduction of a prior serves to “spread out” some of the probability mass over unobserved states, increasing the amount of exploration done for unfamiliar regions of the space. We investigated the tradeoff between sampling in regions that are known to be likely and in new regions. As α grows, the performance of our algorithm first improves, then gradually decreases, as we would expect.

4 Alternative belief states representations

While this approach allows us to generate samples from unexplored parts of the space, it does so blindly: all unsampled states are treated in exactly the same way. However, our state space is not a completely arbitrary set of points. Two states \mathbf{x} and \mathbf{x}' which give the same values to almost all of the variables in our domain may be quite similar, and it may make sense to assume that their probabilities are much closer than that of other pairs. That is, we want to use our results for the states that we sampled to induce the probabilities of other states. This task is precisely a *density estimation* task (a type of unsupervised learning), where the set of sampled states are the training data.¹

As in any learning task, we must first define the hypothesis space. Essentially, our representations above fall into the category of *nonparametric* density estimators [Sco92]. (Roughly speaking, they are a discrete form of Parzen window.) As applied in our setting, these density estimators have no bias (and high variance); thus, they are incapable of generalizing from the training data to the rest of the space. In this section, we explore alternative representations of discrete densities that have higher bias and a correspondingly higher generalization power. As we discussed in the introduction, not every representation is suitable for our needs. Our representation must be significantly more compact than the full joint over the state variables; it must support an effective sampling process; and it must be easily learned. Two appropriate representations are Bayesian networks and *density trees*.

Bayesian networks. Given our overall problem, a Bayesian network representation for our distribution seems particularly appropriate. After all, our process is represented as a DBN, and is therefore highly structured. While it is known that conditional independencies are not maintained in the belief states [BK98], it is reasonable to assume that some of the random variables in a time slice are

¹If we view SOF as doing a process akin to bootstrapping by sampling from its own samples, our extension is akin to *smoothed bootstrapping* [Sil86].

only weakly correlated with each other, and perhaps even weaker when conditioned on a third variable.

There has been a substantial amount of recent work on learning Bayesian networks from data (see [Hec95] for a survey). The simplest option is to fix the structure of the Bayesian network and to use our data to fill in the parameters for it. This process can be accomplished very efficiently, by a simple traversal over our data. Specifically, if our Bayesian network contains a node X with parents \mathbf{Y} , then we need to estimate each of the parameters $P(X = x \mid \mathbf{Y} = \mathbf{y})$. The *maximum likelihood* estimate for these parameters would be $\frac{\sigma_{sc+}(\mathbf{x}, \mathbf{y})}{\sigma_{sc+}(\mathbf{y})}$. However, maximum likelihood estimates result in precisely the type of numerical overfitting (and particularly zero probability estimates) that we strove to avoid in the previous section. It turns out that, if we instead estimate the parameter as $\frac{\sigma_{sc+}(X=x, \mathbf{Y}=\mathbf{y})}{\sigma_{sc+}(\mathbf{Y}=\mathbf{y})}$, we get the effect of introducing a Dirichlet prior over each of our BN parameters. For a given Bayesian network structure B , the resulting distribution $\sigma_{bn(B)}$ is the one that minimizes $D(\sigma_{sc+} \parallel \sigma_{bn(B)})$ among all distributions representable by B .

One potential problem with this approach is that the BN structure B must be determined in advance, based on some prior knowledge of the user or on a manual analysis of the DBN structure. Furthermore the BN structure is fixed over the entire length of the run, whereas the true belief state $\sigma^{(t)}$ can change drastically as the process evolves. It seems quite likely that the most appropriate BN structure for approximating $\sigma^{(t)}$ also varies. This observation suggests that we select a different BN structure for each time slice. Unfortunately, learning of BN structure is a hard problem. Theoretically, even the problem of learning the optimal structure where each node is restricted to have at most k parents is NP-hard for any $k > 1$ [CHG95]. Pragmatically, the algorithms for this learning task are expensive, performing a greedy search, with multiple restarts, over the combinatorial (and superexponential) space of BN structures.

One option is to restrict our search to tree-structured BNs—ones where each node has at most one parent. Chow and Liu [CL68] present a simple (quadratic time) algorithm for finding the tree-structured BN whose distribution is closest—in terms of relative entropy—to the one in our data. The intuition is that, in a tree-structured BN, the edges should correspond to the strongest correlations. Thus, the algorithm introduces a direct connection between the variables whose *mutual information* [CT91] is largest. Formally, for each pair of variables X_i, X_j , we define an *edge-weight*

$$W(X_i, X_j) = \sum_{x_i, x_j} \sigma_{sc+}(x_i, x_j) \log \frac{\sigma_{sc+}(x_i, x_j)}{\sigma_{sc+}(x_i)\sigma_{sc+}(x_j)}$$

which is precisely the mutual information between X_i and X_j in σ_{sc+} . We then choose a maximum-weight spanning

tree over these nodes, where the weight of the tree is the sum of the weights of the edges it spans. We then select an arbitrary root for the tree, and fill in the conditional probability tables for the nodes in the network using σ_{sc+} (as in the case of a fixed BN). Here, again, choosing the parameters using σ_{sc+} is equivalent to introducing a Dirichlet prior over the network parameters. Chow and Liu show that the distribution σ_{st} represented by the resulting spanning tree minimizes $D(\sigma_{sc+} \parallel \sigma_{st})$.

Density trees. A *discrete density tree* is similar in overall structure to a classification decision tree. However, rather than representing a conditional distribution over some distinguished class variable given the features, the **density tree** represents a probability distribution σ_{dt} over some set of variables \mathbf{X} . Each interior node in the tree is labelled with a variable X , and the branches at that node with values x for that variable. A path on the tree to a node n thus corresponds to an assignment \mathbf{y}_n to some subset of the variables in the domain \mathbf{Y}_n .

The tree is a recursive representation of a multivariate distribution. At a high level, the tree structure partitions the space into bins, corresponding to the leaves in the tree. The distribution at each leaf n is uniform over the variables $\mathbf{X} - \mathbf{Y}_n$; the different leaf distributions are combined in a weighted average, where the weight of a leaf is simply the product of the edge-weights along the path to it. More technically, letting \mathbf{Z}_n represent $\mathbf{X} - \mathbf{Y}_n$, we have that if n is a leaf, then $\sigma_{dt}(\mathbf{Z}_n \mid n)$ is uniform over the values of \mathbf{Z}_n ; if n is an interior node labelled with X , then $\sigma_{dt}(\mathbf{Z}_n \mid n) = \sum_x \sigma_{dt}(X = x \mid n) \cdot \sigma_{dt}(\mathbf{Z}_n - \{X\} \mid n_x)$, where n_x is the child of n corresponding to the value x of X and $\sigma_{dt}(X = x \mid n)$ is the weight along the edge to it.

Our error function for the density tree learning algorithm is the relative entropy between our empirical distribution σ_{sc+} and σ_{dt} . We use a greedy algorithm which is very similar to the one for classification trees. We start out with the tree containing only the root node. We then iteratively split nodes on the variable that most decreases this error function. At each point, we estimate the parameters using σ_{sc+} , as we did for BNs. We use a greedy algorithm to determine the splits. The contribution that n makes to the overall relative entropy, if it remains a leaf, is proportional to $D(\sigma_{sc+}(\mathbf{Z}_n \mid \mathbf{y}_n) \parallel u_{\mathbf{Z}_n})$, where $u_{\mathbf{Z}_n}$ is the uniform distribution over the assignments z to \mathbf{Z}_n . If we split n on a variable X , each of its children n_x (assuming they remain leaves) would make a contribution proportional to $\sigma_{sc+}(x \mid \mathbf{y}_n) D(\sigma_{sc+}(\mathbf{Z}_n - \{X\} \mid \mathbf{y}_n, x) \parallel u_{\mathbf{Z}_n - \{X\}})$. It is easy to show that the decrease in the relative entropy is precisely $D(\sigma_{sc+}(X \mid \mathbf{y}_n) \parallel u_X)$. We split n on that variable X which maximizes this decrease. Intuitively, this rule makes perfect sense: if we are representing the distributions at the leaves as uniform, then we should first extract these variables whose marginal distribution at n is the farthest from being uniform.

In order to avoid overfitting, we prevent the density-tree

	Relative error	#samples/slice	runtime/slice (minutes)
counting	$2.275 \pm 1.07 \times 10^{-4}$	1024 ± 1066	0.722
Chow-Liu tree	$2.106 \pm 0.75 \times 10^{-4}$	962 ± 977	0.044
BN 1 (29 params)	$2.102 \pm 0.96 \times 10^{-4}$	981 ± 970	0.064
BN 2 (340 params)	$2.104 \pm 0.79 \times 10^{-4}$	962 ± 1045	0.07
BN 3 (1401 params)	$2.112 \pm 0.73 \times 10^{-4}$	990 ± 1045	0.07
density tree	$1.816 \pm 0.89 \times 10^{-4}$	985 ± 1063	0.068

Figure 4: Means and standard deviations for different belief state representations

from growing to fit all of the samples. We utilize the standard idea of *early stopping*; our stopping rule prevents a node from splitting when the improvement to the relative entropy score is lower than some minimal amount. Specifically, we only allow a split of n on X when $\sigma_{sc+}(\mathbf{y}_n) \cdot D(\sigma_{sc+}(X \mid \mathbf{y}_n) \parallel u_X)$ is higher than some threshold.

We note that our notion of a density tree draws upon the literature of semiparametric density estimation techniques for continuous densities [Sco92]. The uniform distribution over samples at each leaf is similar to multidimensional histogram techniques; however, the tree structure allows variable-sized bins, and therefore greater flexibility in matching the number of parameters to the complexity of the distribution.

5 Experimental results

To provide a more realistic comparison, we tested the different variants of our algorithm on the practical WATER DBN [JKOP89], used for monitoring the biological processes of a water purification plant. (Comparable results were obtained for the CAPITAL network.) The WATER DBN had a substantially larger state space, with 27,648 possible values taken by the (non-evidence) variables. The structure of the WATER network is shown in Figure 1(c).

We experimented with several belief state representations: simple counting (SOF extended with priors); three different Bayesian networks of fixed structure, with 29, 340, and 1401 parameters respectively; Chow-Liu spanning trees; and density trees. We tested each representation on 10 runs, each of length 100, and where we used a variable-samples approach with a target weight of 5. For each run, we tested the average relative entropy error over the run.² (We also tested L_1 error, with comparable results.) We then computed the mean and standard deviation of these run-average errors for the different representations. We did the same for the number of samples utilized per time slice. The results are shown in Figure 4.

Not surprisingly, the worst performer in terms of ac-

²We note that the momentary errors within a run—for belief states at individual time slices—can also vary widely, as can be seen from Figure 3. We tested the standard deviation of the momentary errors within a run, and it was approximately the same among all representations—around 50–55% of the overall average for the run. We omit the detailed results.

curacy is the simple counting approach. The performance of the Chow-Liu trees and the fixed Bayesian networks is about comparable, although the Bayesian network with a large number of parameters performs slightly worse than the rest. The density tree approach performs best, with a fairly significant margin. The number of samples used by the different approaches are not significantly different. What is significant is the fact that the number of samples generated is a factor of 15–25 smaller than the number of states in the state space. Indeed, the running times for the different approaches are all significantly lower than the 1.89 minutes per time slice required by exact inference. We note that the running times were all estimated on simple prototype code. We expect the running times for optimized code to be significantly lower. However, the relative efficiencies of the different algorithms should remain the same.

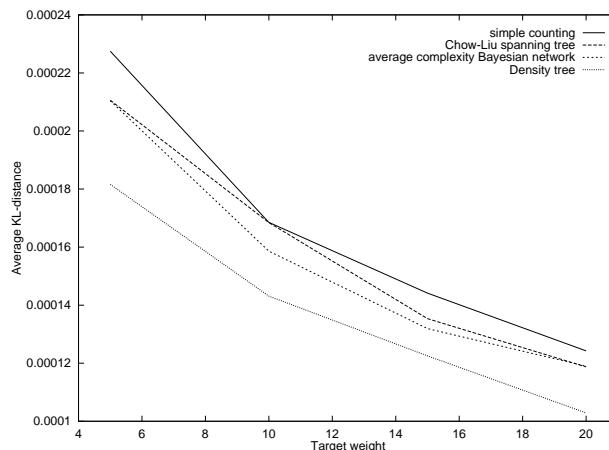


Figure 5: Average error for the WATER network for different target weights. The average is over 10 runs of 100 time slices each.

Figure 5 gives more evidence in favor of the density tree approach, demonstrating that it makes somewhat better use of data. The graph is a type of learning curve for the different approaches: the average error as a function of the target weight. We see that, for any given target weight, the density tree achieves higher accuracy. Furthermore, as we increase the target weight for our sampling algorithm, the error in the density tree approach decreases slightly faster. We note that this improvement does not come at the expense of increasing the overall number of samples: our experiments show that the average number of samples used is essentially identical for the different algorithms, and essentially linear in the target weight.

We believe that two factors contribute to making density trees a suitable representation for this task. The first is its inductive bias. A BN representation reflects an assumption that some of the random variables in the domain influence each other only weakly or indirectly via other variables. A density tree representation reflects an assumption that the distribution is substantially different when condi-

tioned on different values of the same variable. Our results indicate that the variability across different values of a variable is a more significant factor than any (weak) independences found in the distribution. We believe that the evidence serves to sharply skew the distribution in a certain direction, making it much more important for the approximate probability distribution to appropriately model that part of the space. Indeed, an examination of the trees produced by the density tree algorithm for different time slices shows that the parts of the tree corresponding to more likely parts of the space are usually represented using a much finer granularity—with subtrees that are two or more levels deeper—than the less likely ones.

A secondary factor that we believe also contributes to these performance results is the more flexible choice of the structure of the representation. This flexibility, which is shared by Chow-Liu trees and density trees, allows the representation of the approximate belief state to adapt to the current state of the process. An examination of the actual models learned by these algorithms at different points in time, shows that the structure does, in fact, vary significantly. This property is particularly helpful in the density tree case, as the most likely part of the state space changes in virtually every time slice.

6 Extensions and Conclusions

This paper deals with sampling-based approximate monitoring algorithms for a stochastic dynamic process. We have proposed the use of machine learning techniques in order to allow the algorithm to generalize from samples it has generated to samples it has not. We have shown that this idea can significantly improve the quality of our tracking for a given allocation of computational resources. We note that a related idea [BD97] has been proposed in the domain of combinatorial optimization algorithms, and has proved very effective. There, rather than maintaining a population of candidate solutions (as in genetic algorithms), the “good” candidate solutions generated by the algorithm are used to learn a distribution, from which samples are then generated for the next optimization phase.

We have investigated the use of several representations for our probability distributions. We saw that we get significant benefits from allowing the structure of the distribution to vary according to context—both for different parts of the space within the same distribution, and for different distributions over time. In our density tree representation, this flexibility was part of the definition. It would be interesting to see whether we could get even better performance by allowing the other representations to be more flexible. One possibility is to combine Bayesian networks and density trees; there are several ways of doing so, which we are currently investigating. We are also considering the use of other (computationally more expensive) representations of a density, e.g., as a mixture model where the mixture components have independent features [CS95].

It is interesting to also compare our approach to other types of algorithms for inference in stochastic processes. As we have shown, the number of samples generated by our algorithm is significantly lower than the number of states in the explicit representation of the belief state. Thus, our algorithm allows us to deal with domains in which exact inference is intractable. Another option is to use non-stochastic approximate inference algorithms [GJ96, BK98]. The approach of [GJ96] is not really intended for real-time monitoring, and is probably too computationally expensive to be used in that role. It also applies only to a fairly narrow class of stochastic models. The algorithm of [BK98] is more comparable to ours; essentially, it avoids the sampling step, directly propagating a time t approximate belief state to a time $t + 1$ approximate belief state. For certain types of processes, this approach probably dominates ours, as it avoids the additional variance introduced by the sampling phase. However, it is not obvious how it can be implemented effectively for all belief state representations (e.g., for density trees). Furthermore, it does not apply to processes where the representation of the process itself does not admit exact inference (e.g., highly-connected DBN models or models involving continuous variables).

By contrast, we note that our ideas are not specific to DBNs. The only use we made of the DBN model is as a representation from which we can generate random samples. We believe that our ideas apply to a much wider range of processes. Indeed, Isard and Blake [IB96] have obtained impressive results by using a stochastic sampling algorithm identical to simple SOF for the task of monitoring object motion in cluttered scenes. Here, the process is described using fairly complex continuous dynamics, that do not permit any exact inference algorithm. We believe that our ideas can also be used to provide improved algorithms for complex processes such as these, as well as for processes involving both continuous and discrete variables.

Acknowledgements

We would like to thank Eric Bauer and Xavier Boyen for allowing us to build on their code in our experiments. We would also like to thank Wray Buntine and Nir Friedman for useful comments and references. This research was supported by ARO under the MURI program “Integrated Approach to Intelligent Systems”, grant number DAAH04-96-1-0341, by ONR grant N00014-96-1-0718, and through the generosity of the Powell Foundation and the Sloan Foundation.

References

- [Ast65] K. J. Aström. Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anal. Applic.*, 10:174–205, 1965.
- [BD97] S. Baluja and S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proc. ICML*, 1997.
- [BK98] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. UAI*, 1998. To appear.
- [CAL94] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.
- [CHG95] D.M. Chickering, D. Heckerman, and D. Geiger. Learning Bayesian networks: Search methods and experimental results. In *Proc. AI & Stats*, pages 112–128, 1995.
- [CL68] C.K. Chow and C.N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transaction on Information Theory*, IT-14:462–467, 1968.
- [CS95] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In U. Fayyad et al., editor, *Advances in Knowledge Discovery and Data Mining*. AAAI Press, 1995.
- [CT91] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.
- [Deg86] M.H. Degroot. *Probability and statistics*. Addison-Wesley, 1986.
- [DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Comp. Int.*, 5(3), 1989.
- [DL97] P. Dagum and M. Luby. An optimal approximation algorithm for Bayesian inference. *Artificial Intelligence*, 93(1–2):1–27, 1997.
- [FHKR95] J. Forbes, T. Huang, K. Kanazawa, and S.J. Russell. The BATmobile: Towards a Bayesian automated taxi. In *Proc. IJCAI*, pages 1878–1885, 1995.
- [GJ96] Z. Ghahramani and M.I. Jordan. Factorial hidden Markov models. In *NIPS 8*, 1996.
- [Hec95] D. Heckerman. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1995.
- [IB96] M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. In *Proc. ECCV*, volume 1, pages 343–356, 1996.
- [JKOP89] 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, Jüdex Datasystemer A/S, Aalborg, Denmark, 1989.
- [Kja92] U. Kjærulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proc. UAI*, pages 121–129, 1992.
- [KKR95] K. Kanazawa, D. Koller, and S.J. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *Proc. UAI*, pages 346–351, 1995.
- [Sco92] D.W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley, 1992.
- [Sil86] B.W. Silverman. *Density estimation for statistics and data analysis*. Chapman & Hall, 1986.
- [SP89] R. D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Proc. UAI*, 1989.