Space-efficient inference in dynamic probabilistic networks

John Binder, Kevin Murphy, Stuart Russell*
Computer Science Division
University of California
Berkeley, CA 94720

Abstract

Dynamic probabilistic networks (DPNs) are a useful tool for modeling complex stochastic processes. The simplest inference task in DPNs is *monitoring* — that is, computing a posterior distribution for the state variables at each time step given all observations *up to* that time. Recursive, constant-space algorithms are well-known for monitoring in DPNs and other models. This paper is concerned with *hindsight* — that is, computing a posterior distribution given both past and future observations. Hindsight is an essential subtask of learning DPN models from data. Existing algorithms for hindsight in DPNs use $O(SN)$ space and time, where N is the total length of the observation sequence and S is the state space size for each time step. They are therefore impractical for hindsight in complex models with long observation sequences. This paper presents an $O(S \log N)$ space, $O(SN \log N)$ time hindsight algorithm. We demonstrates the effectiveness of the algorithm in two real-world DPN learning problems. We also discuss the possibility of an $O(S)$–space, $O(SN)$–time algorithm.

1 Introduction

Dynamic probabilistic networks are variants of probabilistic (Bayesian) networks designed to represent stochastic temporal processes. They were first used extensively by Dean and Kanazawa [1989], and have since become a standard tool in AI. As Figure 1 shows, a DPN consists of infinitely repeated slices; the variables in the t'th slice represent the state of the process at time t. We assume for the sake of expository simplicity that the variables within a slice can be divided into state variables $\mathbf{X}_t$, which are always hidden, and evidence variables $\mathbf{E}_t$, which are always observable. We also assume that only adjacent time slices are connected, and that the topological structure and conditional distributions are identical for all time slices. That is, the DPN is specified by defining the structure and interconnectivity of the first two slices, and this is then "unrolled" N times, to provide storage space to hold the $\mathbf{X}_t$ and $\mathbf{E}_t$ values for $t \in [0, N]$.

If there is only one hidden variable and one observable variable per time slice, a DPN is equivalent to a Hidden Markov Model (HMM). The main advantage of DPNs over HMMs arises when the state can be decomposed into several state variables, as in Figure 2. (Thus, Ghahramani and Jordan [1995] refer to DPNs as "factorial HMMs.") If each state variable is directly influenced by at most a constant number of other variables, then the number of parameters required to specify the DPN will be linear in the number of state variables, whereas for standard HMMs it is exponential. This reduction makes both inference and learning potentially much more efficient.

This paper begins with a discussion of the DPN *monitoring* task—that is, computing a posterior distribution for the state variables at each time step given all observations *up to* that time. Recursive, constant-space algorithms are well-known for monitoring in DPNs and other models. We then discuss the problem of *hindsight*—that is, computing a posterior distribution given both past and future observations. Hindsight is an essential subtask of learning DPN models from data. Existing algorithms for hindsight — which simply unroll the DPN and treat it like a static network — use $O(SN)$ space and time, where N is the total length of the observation sequence and S is the state space size for each time step.

*This research was funded by the National Science Foundation under grant no. FD96-34215, and by ARO under the MURI program "Integrated Approach to Intelligent Systems," grant number DAAH04-96-1-0341.

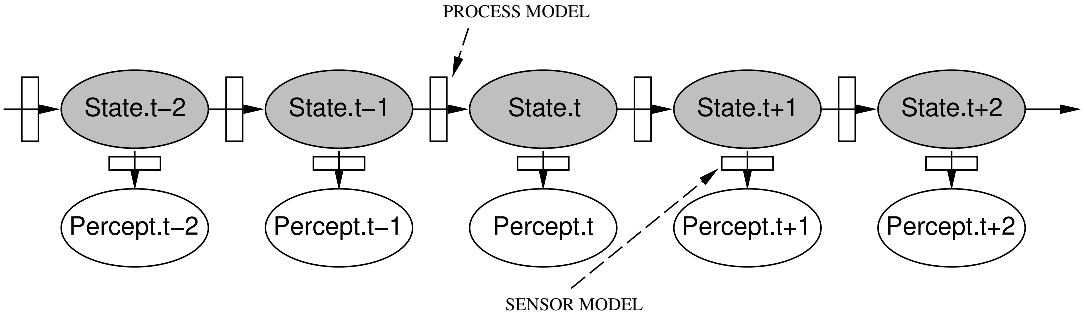

Figure 1: The generic structure of a dynamic probabilistic network, showing the state variables $\mathbf{X}$ and the evidence variables $\mathbf{E}$ schematically. The *sensor model* describes $P(\mathbf{E}_t|\mathbf{X}_t)$ and the *process model* describes $P(\mathbf{X}_{t+1}|\mathbf{X}_t)$.

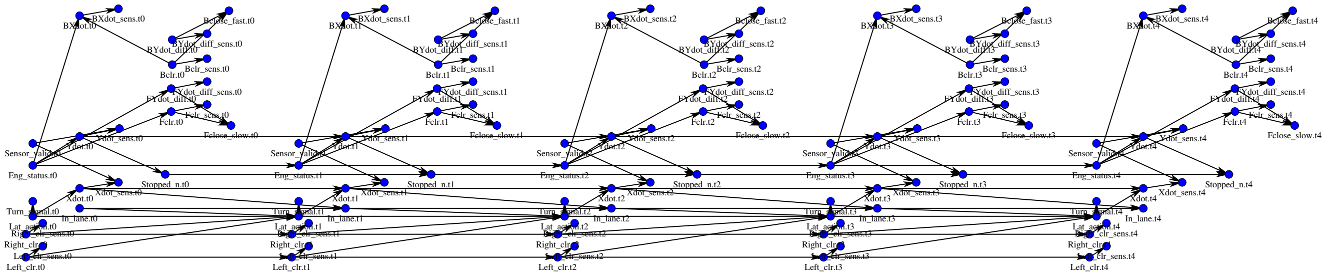

Figure 2: A fragment of an actual DPN, showing the internal structure of each time slice. This was used to model the behavior of cars driving on the freeway [Forbes *et al.*, 1995].

They are therefore impractical for hindsight in complex models with long observation sequences. As an example of the severity of the space problem, we note that the network in Figure 2 has $S \approx 10^6$ and, during model learning, may require $N \approx 10^5$. Therefore, we present a simple divide-and-conquer algorithm that reduces the space requirements from $O(SN)$ to $O(S \log N)$, thus making such applications feasible in practice. We present experimentals results to validate our claims. We also briefly discuss a possible way of reducing the space requirements even more dramatically, namely to use only $O(S)$ space. This would be particularly useful for online learning algorithms.

2 Monitoring in DPNs

The most common use of DPNs is for monitoring, i.e., computing $P(\mathbf{X}_t|\mathbf{E}_{0:t})$, where we use the notation $\mathbf{E}_{i:j}$ as shorthand for the variables $\mathbf{E}_i, \ldots, \mathbf{E}_j$. In this capacity, a DPN allows one to keep track of the state of a process given partial, noisy observations. For example, one could estimate the progress of a disease in a patient from clinical observations over time, or estimate the position of a missile from a sequence of radar observations. In control theory, this task is called *filtering*. The well-known *Kalman filter* [Kalman, 1960] can be viewed as a special case of a continuous-variable DPN in which the sensor model is restricted to be Gaussian and the process model is restricted to be linear with Gaussian noise [Alag and Agogino, 1996]. We note that DPNs can handle a much larger variety of processes than Kalman filters.

Standard arguments (see e.g., [Russell and Norvig, 1995, p.509]) show that monitoring corresponds to the following recursive update equation

$$P(\mathbf{X}_{t+1}|\mathbf{E}_{0:t+1}) = \alpha P(\mathbf{E}_{t+1}|\mathbf{X}_{t+1}) \times \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1}|\mathbf{X}_t{=}\mathbf{x}_t) P(\mathbf{X}_t{=}\mathbf{x}_t|\mathbf{E}_{0:t}) \quad (1)$$

where α is a normalizing constant. For future reference, we represent this equation more abstractly as

$$\mathbf{F}_{t+1} = \mathrm{FwdOp}(\mathbf{F}_t, \mathbf{E}_{t+1}) \quad (2)$$

where we have introduced the forward probabilities $\mathbf{F}_t = P(\mathbf{X}_t|\mathbf{E}_{0:t})$. We assume the initial "boundary condition" $\mathbf{F}_0$ is known.

The process of computing the forward probabilities can be implemented in a variety of ways. Kjaerulff [1992] describes dHugin, an extension of the Hugin package that handles DPNs via direct manipulation of the join tree. Russell and Norvig [1995] describe the roll-up method, which operates directly on the DPN. Finally, one can implement Equation 1 simply by enumerating over the possible states $\mathbf{x}_t$.

The space and time complexity of monitoring can be expressed in terms of N, the total number of time steps; $|\mathbf{E}|$, the number of evidence variables per time step; $|\mathbf{X}|$, the number of state variables per time step; and S, the space required to store the probability distribution over

$\mathbf{X}_t$. In the worst case, $S = 2^{|\mathbf{X}|}$, which corresponds to a completely connected network, even if the original DPN was only sparsely connected. This is because conditioning on past evidence causes variables in the current slice to become directly dependent.[1] Storing all the input ($\mathbf{E}_t$ for all t) and output ($\mathbf{F}_t$ for all t) takes $O(N|\mathbf{E}|)$ and $O(NS)$ space respectively. However, it is reasonable to assume that the input is available as a stream from the environment or is read in sequentially from secondary storage, and similarly for the output. That is, we assume that there are "producer" and "consumer" processes for the input and output; we just focus on the amount of temporary working space required by the montoring algorithm itself, namely $O(S)$. The time required is of course $O(SN)$.

The related task of *prediction*, computing $P(\mathbf{X}_t|\mathbf{E}_{0:N})$ for $t > N$, can be performed by running a monitoring algorithm with no evidence beyond N, and hence also requires $O(S)$ working space.

3 Hindsight in DPNs

In some settings, we may wish to take "future" evidence, as well as past and present evidence, into account e.g., where the t index does not refer to time, but is simply an index into a static sequence. The observations after t may help to eliminate uncertainty about the state at t. For example, one can deduce who was in the house at the time of the murder by subsequent observation to see who leaves the house, even if one neglected to observe the house prior to the murder. The process of computing $P(X_t|\mathbf{E}_{0:N})$ for $t \in [0 \ldots N]$ is called *smoothing* in control theory, where it is often used to reduce the apparent wiggliness in a trajectory computed by filtering. In AI, perhaps the most important use of hindsight is in learning. In order to learn a DPN model from observation sequences, it is necessary to compute likelihoods for the hidden variables given *all* available data [Lauritzen, 1995; Russell *et al.*, 1995], hence hindsight is an integral part of DPN learning.

The most obvious algorithm for hindsight is to perform monitoring in the forwards and backwards directions, and to combine the information at each time step, as follows:

$$\begin{aligned} & P(\mathbf{X}_t|\mathbf{E}_{0:t}, \mathbf{E}_{t+1:N}) \\ & \quad = \alpha P(\mathbf{X}_t|\mathbf{E}_{0:t}) P(\mathbf{E}_{t+1:N}|\mathbf{X}_t) \\ & \quad = \mathrm{CombineOp}(\mathbf{F}_t, \mathbf{B}_t) \end{aligned} \tag{3}$$

where we have introduced the combination operator and the quantity $\mathbf{B}_t = P(\mathbf{E}_{t+1:N}|\mathbf{X}_t)$ to denote the backwards probabilities. These can be updated as follows:

$$\begin{aligned} \mathbf{B}_t \;=\; & \sum_{\mathbf{x}_{t+1}} P(\mathbf{E}_{t+1}|\mathbf{X}_{t+1}{=}\mathbf{x}_{t+1}) \\ & \times P(\mathbf{X}_{t+1}{=}\mathbf{x}_{t+1}|\mathbf{X}_t) \\ & \times P(\mathbf{E}_{t+2:N}|\mathbf{X}_{t+1}{=}\mathbf{x}_{t+1}) \end{aligned} \tag{4}$$

$$= \mathrm{BackOp}(\mathbf{B}_{t+1}, \mathbf{E}_{t+1})$$

Again, we assume the boundary condition $\mathbf{B}_N$ is known. We can think of this method as propagating the forwards and backwards "messages" $\mathbf{F}_t$ and $\mathbf{B}_t$ from both ends towards slice t (which takes $O(SN)$ time and $O(S)$ working space), combining them, and then repeating for each t. Hindsight for the complete sequence therefore requires $O(N^2S)$ time and $O(S)$ working space.

By storing $\mathbf{F}_t$ and $\mathbf{B}_t$ at each time step (i.e., caching intermediate results), we get an algorithm that is essentially identical to Pearl's π-λ message-passing algorithm for chains, to the join-tree algorithm operating on an explicitly represented DPN with N slices, or to the forward-backward HMM algorithm (see also [Smyth *et al.*, 1996]). All these approaches require $O(SN)$ time, but unfortunately require $O(SN)$ space, which is impractical. We now go on to discuss a simple divide-and-conquer algorithm that requires $O(SN \log N)$ time and $O(S \log N)$ space.

4 The space-efficient algorithm

The essential idea of our space-efficient method is to store the $\mathbf{F}_t$ and $\mathbf{B}_t$ messages at $k-1$ intermediate "checkpoints" or "islands", thereby dividing the original sequence into k segments; we then recursively apply the hindsight algorithm to each smaller segment, making use of the boundary conditions stored at each checkpoint. For example, if $N = 24$ and $k = 2$, we compute $\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_{24}$ and $\mathbf{B}_{23}, \mathbf{B}_{22}, \ldots, \mathbf{B}_0$ by iteratively applying the forward and backward operators to the boundary conditions $\mathbf{F}_0$ and $\mathbf{B}_{24}$ respectively, but we only store $\mathbf{F}_t$ and $\mathbf{B}_t$ for $t = 12$. We compute $P(\mathbf{X}_{12}|\mathbf{E}_{0:N}) = \mathrm{CombineOp}(\mathbf{F}_{12}, \mathbf{B}_{12})$, pass it to the consumer[2], and then recursively apply the hindsight algorithm to the sub-sequences for $t = 1 \ldots 11$, using $\mathbf{F}_0$ and $\mathbf{B}_{12}$ as boundary conditions, and for $t = 13 \ldots 23$, using $\mathbf{F}_{12}$ and $\mathbf{B}_{24}$ as boundary conditions.

The total working space required by this approach is determined by two parameters: k, the number of checkpoints we store at each level, and D, the number of levels of recursion before we invoke the base case algorithm (which uses linear time and linear space). If we treat k as a fixed parameter, and recurse "all the way to the bottom" (i.e., apply the base case only to segments of length 1), we have $D = \log_k N$. The working space is then $O(kS \log_k N)$, since we store k checkpoints, each of size $2S$, at each level of recursion, and there are $D = \log_k N$ levels of recursion. The time required by this approach, $T(N)$, is the time taken to propogate the messages across k segements of length roughly N/k each, plus the time to solve the k subproblems: $T(N) = k(N/k) + kT(N/k)$, so $T(N) = N \log_k N$. We can decrease the running time, and increase the space requirements, either by increasing k from 2 to N, or by decreasing D from $\log_k N$ to 1.

[1] Technically, the joint distribution has only the independences present in the stationary distribution of the Markov chain represented by the DPN.

[2] The consumer might store $P(\mathbf{X}_{12}|\mathbf{E}_{0:N})$ on secondary storage, or it might use it in a learning algorithm to perform an in-place update of the model parameters, before discarding it.

An interesting compromise is $k = \sqrt{N}$, which takes only twice as long as the standard algorithm and needs only $O(S\sqrt{N})$ space.

5 Experimental results

We have implemented the abstract operators FwdOp, BackOp, and CombineOp in terms of a modified version of the Jensen join tree algorithm [Jensen *et al.*, 1990]. The details will be presented in another paper, but the basic idea is to modify the triangulation heuristic to ensure that the resulting join tree has a repeating structure. This repeating block may span two slices in the original network (i.e., it might contain cliques which have nodes from adjacent slices), so that $\mathbf{F}_t$ and $\mathbf{B}_t$ now refer to the forwards and backwards messages associated with the t'th repetition of this block, rather than the t'th time slice.

In Figure 4, we show how much space is required to do inference on the simple network shown in Figure 3 as a function of the length of the sequence. It is clear that by increasing the depth of recursion, D, we can reduce the space requirements dramatically. The time taken for $D = 1$ with $N = 100$ was 1.5 seconds, for $D = 2$ it was 1.96 seconds, and for $D = 3$ it was 2.6 seconds.[3] In Figure 5, we plot a similar curve for the more complicated network shown in Figure 2.

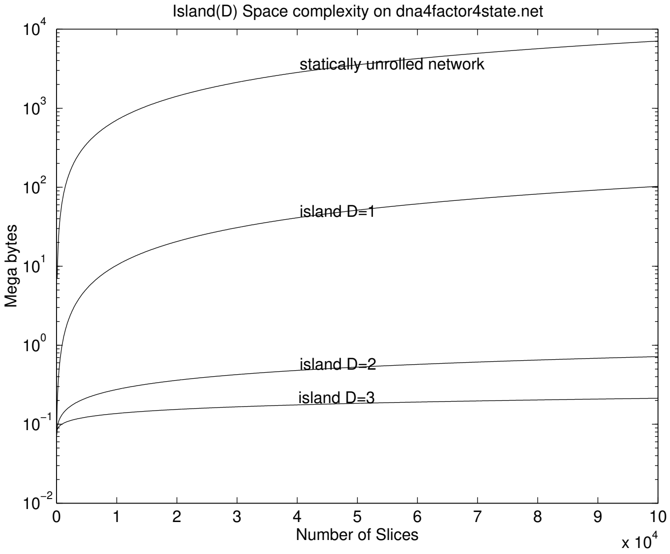

Figure 4: Space complexity of the Island Algorithm on the DNA network. The curve labelled "statically unrolled network" refers to the results produced by applying the standard linear time, linear space algorithm to the network which has been unrolled a fixed number of times. On compiling the DNA network, the size of the repeating clique set, $\mathbf{S}$, is $70,784$ bytes ($17,696$ real numbers) and the size of the $\mathbf{F}$ and $\mathbf{B}$ messages are each 1024 bytes (256 real numbers).

6 Further work: Constant space?

We have shown how to avoid the otherwise crippling space requirements for hindsight in DPNs, giving an algorithm whose space complexity grows logarithmically with the length of the sequence. This has allowed us to address far more complex DPN learning problems than previously possible.

An analogous technique could be applied to cope with cases where S (and not just N) is very large. The idea would be to store check points for only some of the nodes within a slice. For example, in the join tree algorithm, instead of storing messages at every node in the repeating tree structure, we could store messages at only some of them, recomputing the others on demand.[4] This could of course be combined with the current approach to yield an $O(\log N \log S)$-space algorithm.

A more ambitious goal would be to find a constant-space algorithm, that is, one having space requirements which are independent of N. This would be particularly useful in the context of online learning, in which a continuous stream of evidence is arriving (and hence N grows indefinitely). We currently only know how to do this in restricted circumstances. The idea is as follows. According to Equation 4, we can compute the posterior at time step t given $\mathbf{F}_t$ and $\mathbf{B}_t$. We can filter forward all the way to the end to obtain $\mathbf{F}_N$, and $\mathbf{B}_N$ is given. By backward monitoring, we can now compute $\mathbf{B}_{N-1}$. If we can also compute $\mathbf{F}_{N-1}$ from $\mathbf{F}_N$, we are in business. This can be done by *inverting* the process model and by undoing the effects of $\mathbf{E}_N$. We then continue this process all the way back to the beginning of the sequence, computing posteriors as we go.

The details of the inversion process are best understood in terms of matrix operations, where we consider $\mathbf{F}_t$ as a vector of probabilities over all possible states $\mathbf{x}_t$ of the variables in $\mathbf{X}_t$. Let $\mathbf{O}_t$ be the diagonal matrix whose entries are the probabilities in $P(\mathbf{E}_t|\mathbf{X}_t)$, and let $\mathbf{M}$, the transition matrix, contain the probabilities $P(\mathbf{X}_{t+1}|\mathbf{X}_t)$ for all values $\mathbf{x}_{t+1}$ and $\mathbf{x}_t$. (Note that according to our assumptions, $\mathbf{M}$ is independent of time.) Equation 1 now becomes the vector equation

$$\mathbf{F}_{t+1} = \alpha \mathbf{O}_{t+1} \mathbf{M} \mathbf{F}_t .$$

From this we can obtain the inverse operation:

$$\mathbf{F}_t = \alpha' \mathbf{M}^{-1} \mathbf{O}_{t+1}^{-1} \mathbf{F}_{t+1}$$

Thus, in the generic case where $\mathbf{M}$ and $\mathbf{O}$ are invertible, we have a constant-space hindsight algorithm. This algorithm will fail, however, if either matrix is singular. $\mathbf{O}_t$ will be singular only if some entries in $P(\mathbf{E}_t|\mathbf{X}_t)$ are zero, that is, if the observations rule out some of the possible states. $\mathbf{M}$ will be singular if, for example, two columns are identical—which could easily occur if transitions are independent of one of the state variables. Thus,

[3] Experiments were run on a Pentium Pro 200MHz PC with 64 Megabytes of RAM, using Linux and gcc.

[4] The key requirement is that the set of nodes at which we choose to store checkpoint information d-separate the evidence.

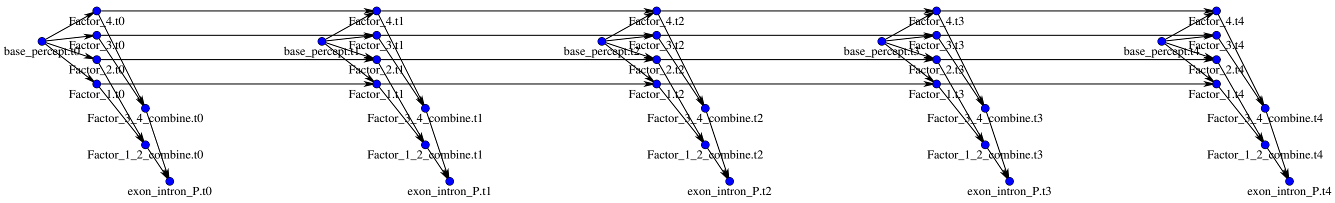

Figure 3: A simple network for learning intron/exon coding behavior in DNA.

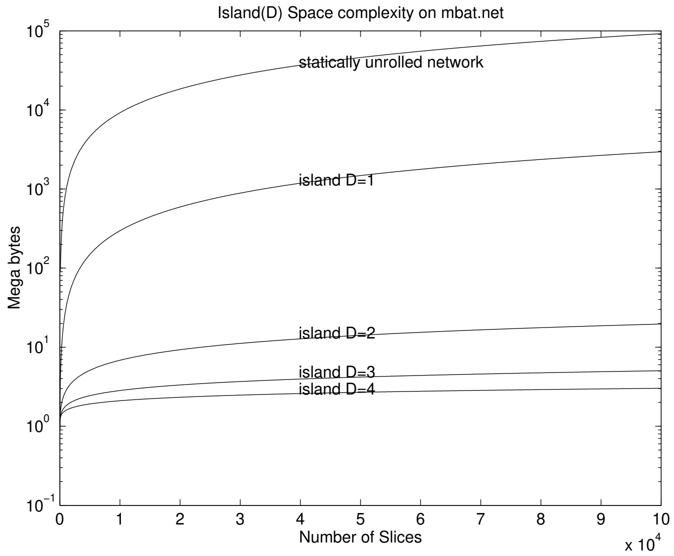

Figure 5: Space complexity of the Island Algorithm on the car network. **S** is 920, 124 bytes (230, 031 real numbers) and the size of the **F** and **B** messages are each 29, 568 bytes (7, 392 real numbers).

invertibility will fail in many realistic situations where information is lost as the process proceeds, and in many others the inversion step will be highly ill-conditioned. It remains to be seen whether it is possible to somehow store a small amount (independent of N) of extra information to make the transformation invertible.

Acknowledgments

Thanks to Paul Horton, Geoff Zweig, Nir Friedman, and the reviewers for useful comments.

References

[Alag and Agogino, 1996] S. Alag and A. Agogino. Inference using message propagation and topology transformation in vector gaussian continuous networks. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 20–27, Portland, Oregon, 1996. Morgan Kaufmann.

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

[Forbes *et al.*, 1995] Jeff Forbes, Tim Huang, Keiji Kanazawa, and Stuart Russell. The BATmobile: Towards a Bayesian automated taxi. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada, August 1995. Morgan Kaufmann.

[Ghahramani and Jordan, 1995] Z. Ghahramani and M. I. Jordan. Factorial hidden Markov models. Technical Report 9502, MIT Computational Cognitive Science Report, 1995.

[Jensen *et al.*, 1990] Finn V. Jensen, Steffen L. Lauritzen, and Kristian G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 5(4):269–282, 1990.

[Kalman, 1960] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, pages 35–46, March 1960.

[Kjaerulff, 1992] U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, pages 121–129, 1992.

[Lauritzen, 1995] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201, 1995.

[Russell and Norvig, 1995] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, New Jersey, 1995.

[Russell *et al.*, 1995] Stuart Russell, John Binder, Daphne Koller, and Keiji Kanazawa. Local learning in probabilistic networks with hidden variables. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1146–52, Montreal, Canada, August 1995. Morgan Kaufmann.

[Smyth *et al.*, 1996] P. Smyth, D. Heckerman, and M. Jordan. Probabilistic independence networks for hidden Markov probability models. Technical Report MSR-TR-96-03, Microsoft Research, Redmond, Washington, 1996.